RL-TR-97-76
Final Technical Report
August 1997

AN ARCHITECTURE FOR EXTERNALLY CONTROLLABLE VIRTUAL NETWORKS AND ITS EVALUATION ON NYNET

Columbia University

Mun Choon Chan, Aurel A. Lazar, and Rolf Stadler

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-76 has been reviewed and is approved for publication.

APPROVED: Bradley J. Harnish

BRADLEY J. HARNISH
Project Engineer

FOR THE DIRECTOR: John Graniero

JOHN A. GRANIERO, Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
August 1997

3. REPORT TYPE AND DATES COVERED
Final Jun 94 - Nov 96

4. TITLE AND SUBTITLE
AN ARCHITECTURE FOR EXTERNALLY CONTROLLABLE VIRTUAL NETWORKS AND ITS EVALUATION OF NYNET

5. FUNDING NUMBERS
C - F30602-94-C-0150
PE - 62702F
PR - 4519
TA - 22
WU - 28

6. AUTHOR(S)
Mun Choon Chan, Aurel A. Lazar, and Rolf Stadler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Columbia University
Center for Telecommunications Research
801 Schapiro Research Building
New York, NY 10027-6699

8. PERFORMING ORGANIZATION REPORT NUMBER
CTR 469-97-03

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/C3BC
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-97-76

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Bradley J. Harnish/C3BC/(315) 330-1884

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The goal of this work was to develop a communication architecture that allows the creation of a virtual enterprise network for a geographically distributed corporation that is made up of a private high-speed backbone and islands of private component networks connected using a public broadband infrastructure. The management and control system of the enterprise network is, as far as possible independent of the characteristics of the underlying public network services and it guarantees quality of service and survivability end-to-end. It allows reconfiguration of the virtual enterprise network during operational time in response to external conditions through dynamic bandwidth allocation on different network control layers, which operate on different time scales. A key idea in the proposed architecture is the enhancement of external control through the use of the Virtual Path Group concept in the construction of the virtual enterprise network. The results of this effort are directly applicable to the extension of the Global Grid backbone into theaters of operations.

Validation of the design was based on network emulation, which included executing the behavior of the network components and their interactions under the control of a parallel simulation kernel running on a high-performance computer. The architecture was implemented on a network emulator on two supercomputers located at the Cornell Supercomputer Center. The emulation system was interactively controlled by and visualized on an Open GL-based graphics workstation at Columbia University that was connected to Cornell over high-speed (See Reverse)

14. SUBJECT TERMS
ATM Network, Virtual Private Networks, Global Grid, Network Management, External Network Control, End-to-end QOS, Network Emulation, Network Visualization, Supercomputer, Parallel Simulation Kernel

15. NUMBER OF PAGES
110

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

SF Form 298

13. Abstract (Continuation)

NYNET links. A set of performance management capabilities was also realized for this service, including quality of service management, virtual path management, and priority management.

Table of Contents

1 Introduction

The goal of this work is to develop a communication architecture that allows for the creation of a virtual enterprise network for a geographically distributed large corporation. The enterprise network is made up of a private high-speed backbone and islands of private component networks connected using public broadband infrastructure (Figure 1). The management and control system of the resulting enterprise network is, as far as possible, independent of the characteristics of the underlying public network services and guarantees quality of service and survivability end-to-end. The results of this effort are directly applicable to the extension of the Global Grid backbone into theatres of operations.

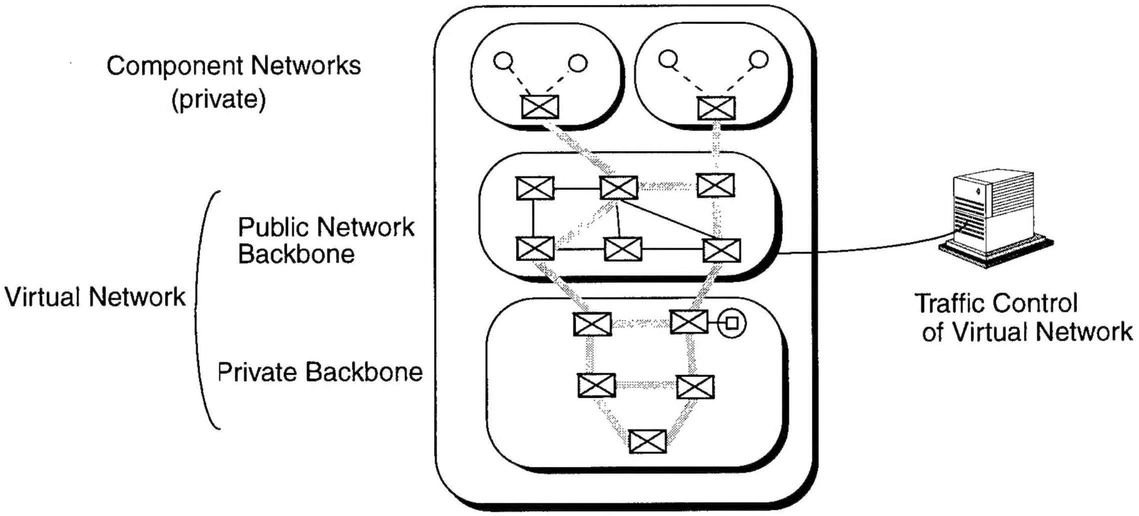


Figure 1 A Virtual network interconnecting a private backbone with isolated component networks.

Specifically, we develop an architectural framework in which we propose solutions to the following questions:

- How should end-to-end quality of service be provided over the private components (private backbone and component networks) and public networks?
- What is the control and management architecture for such a network?
- How should this architecture be evaluated and realized?

Our approach is to construct, on top of a public network environment, a *virtual network* that enables the transparent interconnection of a private backbone with isolated component networks, and that guarantees quality of service. The virtual network allows reconfiguration during operational time in response to external conditions through dynamic bandwidth allocation on different network control layers, which operate on different time-scales. A key idea in the proposed architecture is the enhancement of external control through the use of the virtual path group (VPG) concept in the construction of a virtual private network (VPN). External control is regarded as the same as *customer control* in our context, and the term customer control will be used throughout in the report.

Our approach for validation is based on network emulation, which includes executing the behavior of network components and their interactions under the control of a parallel simulation kernel running on a high-performance machine. The emulation environment allows us to experiment with the functionality and dynamics of virtual networks (and the underlying networks), with greater flexibility and lower cost than implementing components on a real testbed.

We have implemented the architecture on a network emulator first on a KSR-1 and then on a IBM SP2. Both supercomputers are located at the Cornell Supercomputer Center. The emulation system, which can run on either one of the supercomputers, is controlled by and visualized on an Indigo2 workstation at Columbia that is connected to Cornell over high-speed NYNET links. The Network visualization interface is based on OpenGL, a 3D graphics package.

This experimental platform helped us to deliver the proof of concept for our work, and allows us to revise the developed architecture. We are able to validate the functional and dynamic aspects of the key concepts of the architecture for various system configurations. Further, the interactive control and visualization of the architecture enables us to demonstrate the system concepts, especially the concepts of external control and monitoring.

The report is divided into 2 parts: architecture design and prototyping of the proposed architecture. The design of the architecture is described in Chapters 2 and 3. In Chapter 2, a review of existing virtual network services is presented, followed by motivation for external control and description of the proposed architecture. In Chapter 3, we focus on the issues of QOS guarantees and control algorithms. Our approach to evaluation and realization of the proposed architecture is described in Chapters 4 and 5. In Chapter 4, we describe the high performance platforms used for network emulation and the lessons learned. In Chapter 5, we describe the prototyping approach used for architectural evaluation and for bridging the gap between design and realization. Finally, in Chapter 6, we present a summary of our results and possible extensions of our work.

2 VPN Architecture for Customer Control and Management

2.1 Broadband VPN Services

Broadband technology has the potential to change corporate networking in major ways. Broadband networks are aimed at providing quality-of-service (QOS), thus making it possible to support real-time services like voice and video communication, in addition to best-effort data delivery. Due to their ability to integrate different services on the cell-level, they provide a promising platform for distributed multimedia applications that are emerging today. Furthermore, the advent of broadband technology will enable the integration of today's separate corporate networks (voice network, data network), which often rely on different public services (e.g., leased lines for voice traffic and LAN interconnection, frame-relay service for low-volume data exchange) into a single enterprise network, using a single Virtual Private Network (VPN).

A broadband virtual private network (VPN) is a service that provides broadband transmission capability between islands of customer premises networks (CPNs). It is a central building block for constructing a global enterprise network (EN) which interconnects geographically separate CPNs.

A VPN service involves several administrative domains: the customer domain, the domain of the VPN service provider--also called "value added service provider" (VASP)--, and one or more carrier domains [SCH93]. As a result, it is necessary to address the aspects of multi-domain management in the context of VPN service management and provisioning ([HAL95], [LEW95], [TSC95]). The scope of this paper is limited to the customer domain and the interaction between the customer domain and the VPN provider domain.

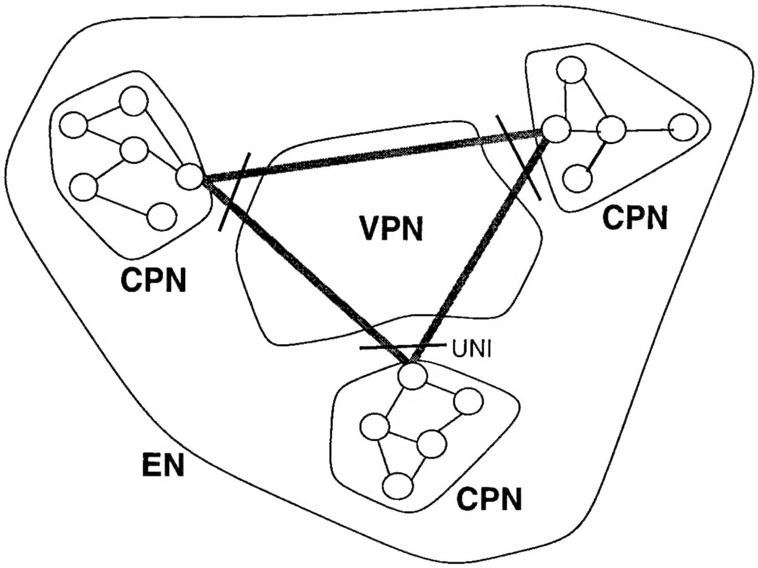


Figure 2 Customer's view of a virtual private network

In the following, we briefly describe different types of broadband VPN services, taking the customer's point of view (Figure 2). Traditionally, leased line circuits based on STM (SDH/ SONET) technology have been used for providing VPN services [YAM91]. The speed of the circuit can be changed by customer-provider cooperative control. However, dynamic bandwidth adjustment for leased line circuits is inefficient and costly compared to ATM-based services, which place no restriction on the line speeds the customer can choose from [HIT94].

Service providers are beginning to offer broadband VPN services using ATM transport networks. Two common approaches are VC-based VPN services ([MOU95], [SAY95], [FOT95]) and VP-based VPN services [ATS93]. These services provide ATM logical links between separate CPNs. In the case of a VC-based VPN service, the customer requests a new VC from the provider for every call to be set up over the VPN. Bandwidth control and management between customer and provider is performed per VC.

In the case of a VP-based VPN service, customers can perform their own call and resource control for a given VP, without negotiating with the VPN provider. Bandwidth control and management between customer and provider is performed per VP.

VC-based and VP-based VPN services replace today's leased line services. They offer customers more flexibility in dynamically requesting adjustments in the VPN capacity. Since networks typically exhibit a dynamic traffic pattern, such a technique of rapid provisioning will result in lower cost for the customer, because pricing is expected to be based on the VPN capacity per time interval allocated to the enterprise network.

A VPG-based VPN service has been proposed to enhance customer control over the VPN [CHA96]. This service is based on the *Virtual Path Group* (VPG) concept, which has been introduced in [HIS94] to simplify virtual path dynamic routing for rapid restoration in a carrier network.

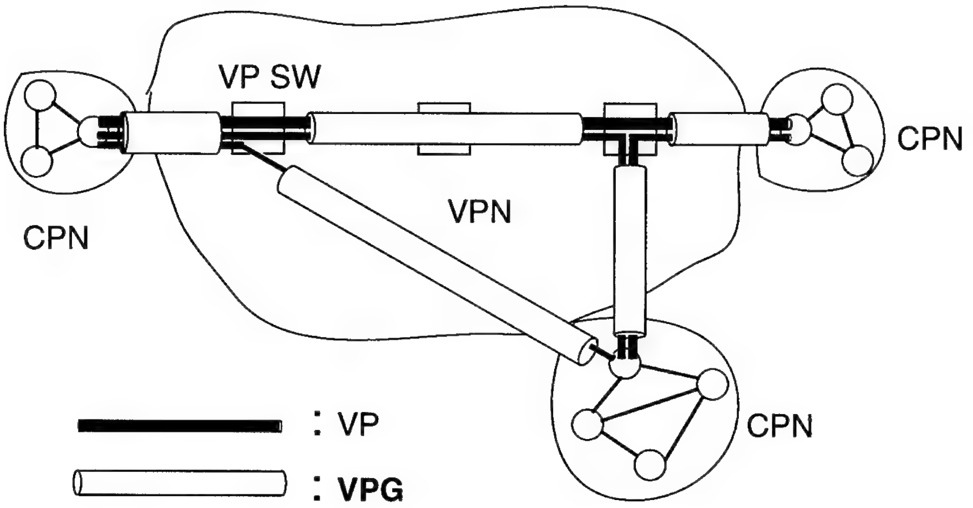


Figure 3 A VPG-based virtual private network

A VPG is defined as a logical link within the public network provider's ATM network. Figure 3 shows a *VPG-based Virtual Private Network* connecting 3 CPNs. A VPG is permanently set up between two VP cross connect nodes or between a VP cross connect node and a CPN switch that acts as a customer access point for the VPN service. Any ATM switch that supports VP switching can be used as a VP cross connect node, including an ATM LAN switch such as FORE ASX-100. A VPG accommodates a bundle of VPs that interconnect end-to-end customer access points. The VPN provider allocates bandwidth to a VPG, which defines the maximum total capacity for all VPs within the VPG. A VPG-based VPN consists of a set of interconnected VPGs.

VPs and VPGs are set up by the network management system of the VPN provider during the VPN configuration phase. Only the network management systems must know about the routes of the VPGs, their assigned bandwidth, and the VPs associated with them. The use of VPGs has no impact on cell switching, as cells are transmitted by VP cross connect nodes based on their VP identifier. In order to guarantee cell-level QOS in the carrier's network, policing functions (Usage Parameter Control) are required at the entrance of each VPG.

The VPG concept enhances the customer's capability for VP capacity control. It allows transparent signalling and dynamic VP bandwidth management within the customer domain. A customer can change the VP capacities, within the limits of the VPG capacities, without interacting with the provider. As a result, the VPG bandwidth can be shared by VPs with different source-destination pairs. Furthermore, customers can independently achieve the optimum balance between the resources needed for VP control and the resources needed to handle the traffic load.

A summary of the three approaches to VPN provisioning is given in Table 1.

Private Network schemes Bandwidth sharing level	ATM leased line [ATS93]	VPN based on cross connect VCs [FOT95]	VPG-based Virtual Private Network [CHA96a]
- bandwidth of a physical link - shared by traffic from different customers	VP capacity control with customer-PN negotiation	——	VPG bandwidth control with customer-PN negotiation
- dedicated bandwidth to a customer - shared by different source-destination(s-d) traffic	——	call by call VC setup by customer	VP capacity control by customer
- dedicated bandwidth to a s-d pair - shared by the same s-d traffic	call by call VC setup by customer		call by call VC setup by customer

Table 1: Summary of Approaches to Broadband VPN Provisioning

2.2 Customer Control

Corporations want to control and manage their enterprise networks according to their own control objectives and management strategies. This implies that a corporate customer, using a VPN service, needs the capability to control and manage its traffic on the VPN--possibly in cooperation with the provider. For the designer of an enterprise network, the question arises, which part of the control functionality is executed in the customer's domain and which part in the provider's domain. More precisely: which functions are performed by the customer alone, which by the provider alone, and which in the form of customer-provider cooperative control.

There are strong reasons for *customer control*, i.e., for running traffic management functions in the customer domain. Firstly, different customers pursue different control and management objectives while running their enterprise networks. For example, customer requirements concerning the traffic carried on in a VPN are very diverse with respect to supporting multimedia traffic with different performance characteristics and performance requirements. Some customers may want to operate a multiclass network with several traffic classes for both real-time and non real-time traffic and a high degree of cell-level multiplexing; others may want to support just one class of traffic with peak rate allocation. Some might want to implement a call priority scheme which enables calls of higher priority to pre-empt those of lower priority when the network is congested; others may want to apply other control schemes in case of congestion. etc., etc. Providers face difficulties to support such diverse requirements. Customers who know their requirements better than the providers may be in a better position to execute control according to their objectives.

Also, operations under customer control can be executed faster than those performed in cooperation with the provider, since no negotiation is required. For example, setting up connections over a VPN can be done by the customer in a distributed way, based only on local information. This allows customers to engineer or configure their traffic control systems in such a way that short connection set-up times can be achieved, which is required by some applications.

Secondly, customers want provider-independent control in order to meet special requirements for the enterprise network [ZER92]. For example, usage collection that permits billing at a level of detail beyond the provider's capability, such as billing at an application level, may be needed. Furthermore, the partitioning of the VPN by the customer may be required to implement sophisticated access control mechanisms, which prevent unauthorized access to certain partitions of the network. Also, automatic fall back mechanisms may be desirable for critical applications that need high network reliability.

Finally, moving the responsibility for VPN traffic management from the provider to the customer accelerates the introduction of Broadband VPN services. Specifically, public VPN services based on CBR VPs can be provided efficiently today [ATS93, FOT95]. However, such a service requires resource control by the customers, since they will be billed based on allocated bandwidth--even if they do not use it.

Obviously, raising the level of customer control increases the complexity of the customer control system. However, recent advances in distributed object-oriented technology make it easier to build network control systems with a rich functionality.

In the next section, we present an architecture for management and control of a broadband VPN service. The architecture is operated by the customer and emphasizes the concept of customer control. We outline how different control and management objectives can be achieved with this architecture. An element of this architecture is the design of a generic resource controller, which can be specialized in order to realize a large class of control schemes, following a customer's specific requirements.

2.3 Customer Control and Management Objectives

From the perspective of traffic control, the customer wants to achieve two sets of objectives. The first set relates to end-to-end QOS requirements for the traffic on the enterprise network, which translates into QOS objectives for the traffic that traverses the VPN. QOS objectives on the cell level are usually expressed in terms of bounds on end-to-end delays and error rates. On the call level, QOS objectives include call blocking constraints and bounds on call set-up times. The second set relates to efficient use of VPN resources, primarily trunk bandwidth.

Efficient use of the VPN bandwidth is very much related to the QOS requirements. On one end of the spectrum is peak rate allocation in the form of CBR VPs, where the provider guarantees upper bounds for delays and loss rates on the VPs [ATS93]. Based on this information, the customers can choose to run their own multiplexing schemes on various levels if more efficiency is desired and less stringent QOS requirements can be tolerated. On the cell-level, exploiting multiplexing among calls with the same source-destination pair in the VPN can be performed using the schemes described in [HYM91, ELW93]. Cell multiplexing among calls with different source-destination pairs can be achieved using the contract region concept [HYM94]. On the call-level, schemes used for VP control (e.g. [OHT92]) can be used to exploit multiplexing among calls with the same source-destination pairs. Finally, the schemes described in [FOT95] and [CHA96a] can be used to multiplex calls with different source-destination pairs. Depending on the type of VPN service the provider offers, the customer can choose to implement a combination of the above described multiplexing schemes in the customer control system.

In terms of managing the enterprise network, customers want capabilities to control the bandwidth cost of the VPN service, define QOS objectives and set preferences and priorities for resource allocation to deal with congestion situations. These management objectives apply to the customer domain only and are different from customer to customer. They define the policies according to which the customer control system operates. Management capabilities can be realized by tuning controllers in the customer control system (Section 4.3). For illustration purposes, we describe below some of the management capabilities we have implemented in our prototype system.

In terms of managing the enterprise network, customers want capabilities to control the cost of the VPN service, define QOS objectives and set preferences and priorities for resource allocation to deal with congestion situations. These management objectives apply only to the customer domain and are different from customer to customer. They define the policies according to which the customer control system operates. Management capabilities can be realized by tuning controllers in the customer control system (see Section 4.3). For illustration purposes, we describe below some of the management capabilities we have implemented in our prototype system.

Cost management allows the customer to define the maximum average cost of the VPN communication resources over a specific period of time. This capability is realized by setting constraints on the negotiation of VPN bandwidth between the customer and the provider. VP management allows the customer to directly manipulate VP bandwidth. Operationally, the control of the VP bandwidth can be executed either automatically by the customer control system or under direct control of the operator of the enterprise network. The operator can allocate a fixed amount of bandwidth to a VP, which must be respected by the control system. QOS and priority management operations define how calls are handled in the enterprise network. In our specific implementation, every call is characterized by a performance class and a priority class. Both classes represent independent concepts. The performance class of a call determines its QOS requirements. QOS management deals with managing the level of service provided to different performance classes. In particular, the customer can modify the blocking objectives of calls belonging to a performance class. The level of priority determines the relative importance of a call. In our scheme, a high priority call can preempt a call of lower priority in case of congestion. The customer can enable and disable priority control and can set blocking objectives for priority classes. The above described management capabilities are orthogonal in the sense that they can be applied independently of one another.

2.4 The Proposed Architecture for Customer Control and Management

In this section, we describe the design of a control system for a VPG-based VPN service. Since a VPG-based VPN service can be seen as an extension of a VP-based VPN service, its control system contains an additional layer of functionality, the VPG control layer, which operates on a medium time-scale.

The customer control system is operated by the customer and runs in the customer domain. In our design, the primary interaction between the customer and the VPN provider relates to negotiation of VPN bandwidth. This gives customers a high degree of control over the traffic that is carried over the VPN. Specifically, customers can perform all aspects of call control and VP control independently of the provider, according to their own objectives and requirements.

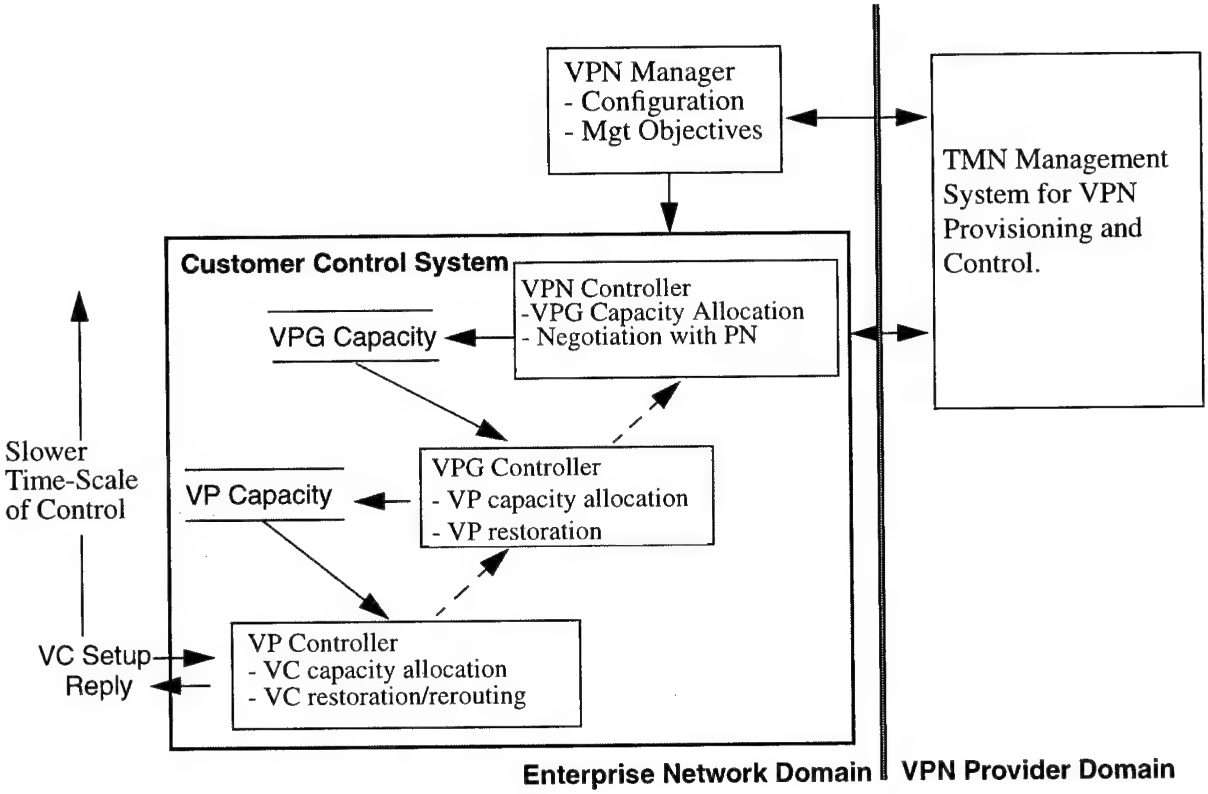


Figure 4 A functional model of the customer control system

Figure 4 shows the systems involved in the provisioning and operation of a VPG-based VPN. In the provisioning phase, information concerning the VPG topology, the VP topology and the mapping between them is exchanged and stored in the management systems of the customer and the provider. Knowledge about the VPGs is also required in the provider's control system, which performs Usage Parameter Control (UPC) per VPG. The use of VPGs has no influence on cell switching and transmission, since cells are switched according to the VP identifiers in their headers.

Figure 4 also shows the organization of the control system according to time-scales. The customer control system contains three classes of controllers: VP controller, VPG controller, and VPN controller. These controllers operate on different time-scales and run asynchronously.

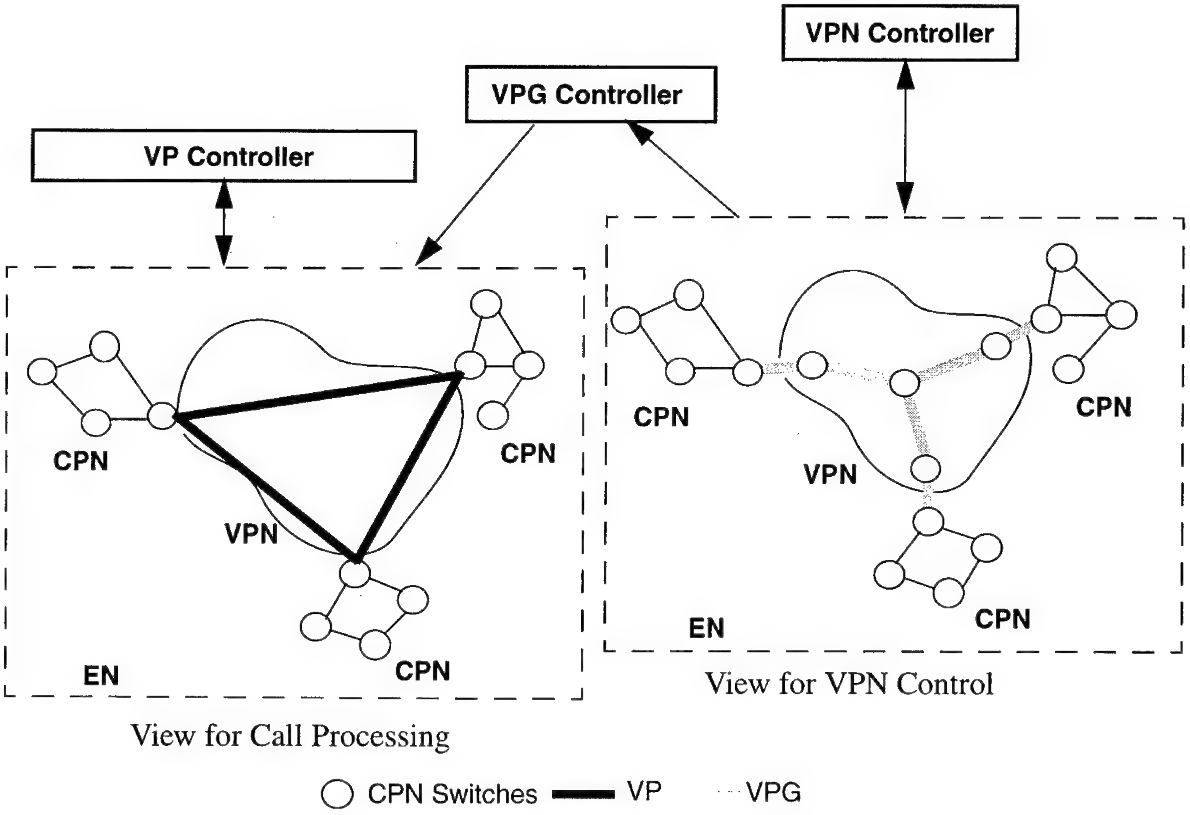


Figure 5 Network views the controllers operate on

We illustrate the interaction among these controllers with an example. Assume that one of the VPs experiences a sudden increase in traffic load. The VP controller that is associated with this VP admits calls as long as there is sufficient capacity. If there is not sufficient capacity available, calls are blocked. On a slower time scale, the VPG controller detects the congestion in this particular VP and attempts to allocate additional bandwidth to this VP. If the increase in traffic load is transient and, therefore, the demand for bandwidth drops after some time, the interaction stops here. Otherwise, if the congestion persists, the VPN controller, which runs on a slower time-scale, will request additional VPN capacity from the provider.

For the purpose of dynamic bandwidth control, a VPG-based VPN can be compared to an ATM network in which the link size can be varied. Therefore, controllers in the customer domain operate on two views of the network (Figure 5). The view on the left side of Figure 5 shows a network of end-to-end VPs which connect a set of CPNs. The view on the right shows a VPG network, which connects the same set of CPNs. The relationship between VPs and VPGs defines the mapping between both views.

The VP controller, which participates in call setup and release in the enterprise network, operates on the left view. The controller decides whether a call can be admitted into the VPN, based on the VP capacity, its current utilization and the admission control policy. The VP controller behaves likes an admission controller. It ensures that enough capacity is available, such that cell-level QOS can be guaranteed for all calls that are accepted. The controller runs on the time scale of the call arrival and departure rates (seconds or below). There can be one VP controller per VP, or one for a set of VPs.

The VPG controller operates on both views. Depending on the state of the VPs (in particular, traffic statistics and VP size) and the control objectives, it dynamically changes the amount of VPG bandwidth allocated to associated VPs. This controller enables customers to exploit variations in utilization among VPs that traverse the same VPG, allowing bandwidth between VPs of different source-destination pairs to be shared without interacting with the provider. In order to guarantee QOS, the sum of the VP capacities must be less than or equal to the capacity of the VPG link. The controller runs on a time-scale of seconds to minutes.

The VPN controller operates on the right view. It is the only controller which interacts with the provider, and it runs on the slowest time scale of all the controllers (minutes or above). The VPN controller dynamically negotiates the bandwidth of the VPG links with the provider, based on traffic statistics and control objectives (e.g., minimizing the VPN cost), while observing the customer's QOS requirements.

2.5 Controller Design

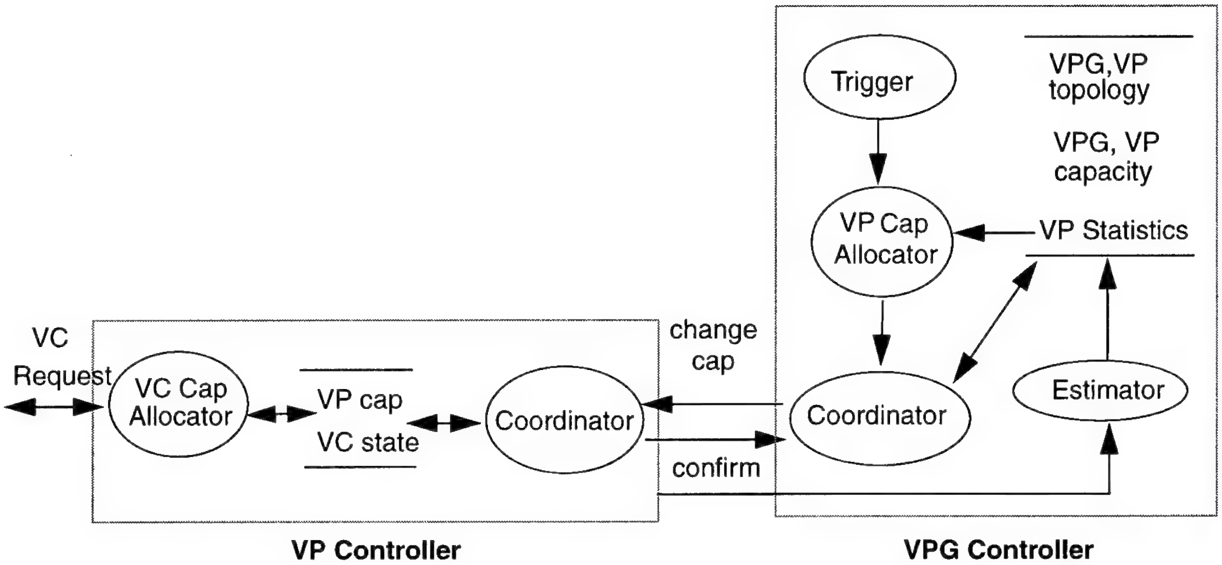


Figure 6 Functional model of a call admission controller interacting with a VPG controller.

Figure 6 shows the functional design of a VP controller and a VPG controller according to our implementation. In this design, the VP controller includes two objects: a VC capacity allocator and a coordinator. The allocator receives requests from a VC connection manager in the customer domain. The coordinator changes the capacity of the VP upon request from the VPG controller. It changes the capacity of the VP only when the bandwidth requirements of the active calls in the VP do not exceed the new capacity.

The VPG controller includes four objects. The trigger object periodically initiates the VP capacity allocator to run the VP allocation algorithm. The coordinator sends the new VP capacities to the coordinators of the associated VP controllers, using a synchronization protocol. Finally, an estimator object collects statistics from the VP controllers. This data is used by the capacity allocator.

Obviously, there exist many ways of realizing the above design, with respect to control algorithms, mechanisms for trigger realization, synchronization protocols, and centralized or distributed implementation of the controllers. For example, the control system may include one VP controller per VP or one centralized controller for the whole VPN. The same applies for VPG control. Also, VP controllers can send bandwidth requests to VPG controllers, triggered by a pressure function, or a VPG controller can periodically recompute the VP capacities and distribute them to VP controllers. Similarly, the synchronization protocols between the VC admission controller and the VPG controller can be realized in different ways. One possibility is that the VP controller, upon receiving a request to change the VP size, checks whether the current utilization is above or below the new size. If the utilization is below, the VP size is changed and a confirmation is sent to the coordinator of the VPG controller. If it is not below, the VP size remains the same and a failure reply is sent instead. In another possible implementation, when the attempt for changing the VP size is not successful, the VP controller waits and blocks further calls from being admitted. Then, the utilization of the VP can only decrease, as calls can leave but no new calls are admitted. When the utilization drops below the new size, the VP size is updated and the reply sent to the VPG controller.

A customer's choice for a specific design of the control system is based upon its control objectives and requirements for the control system, which relate to system size, expected traffic and signalling load, efficiency of resource control and robustness of the control system. In order to enable the realization of a large class of control objectives and control schemes, we have designed a generic controller as one of the building blocks of a customer control system. This generic controller enables many interaction patterns among controllers and is constructed in a modular way.

Figure 6 shows a functional model of the generic controller, which includes two sets of subcontrollers in a symmetrical design. One set of subcontrollers regulates the access to the resource, and the other set controls the size of the resource. The two sets of subcontrollers cooperate by accessing a shared data object, the resource graph. Each set of subcontrollers is made up of three functional components: trigger, allocator, and coordinator. The trigger decides when a computation should be done. The allocator performs the computation, which can be initiated by an external controller or by the trigger. The allocator that controls the access to the resource computes the amount of the resource that should be given to a particular request. The allocator that controls the size of the resource determines the resource capacity. A change of the resource capacity is coordinated by the coordinator object, which facilitates the interaction with other controllers. In particular, it implements the synchronization protocol needed to ensure that state changes among distributed controllers do not violate a set of resource constraints. The resource graph is modeled as two sets of weighted graphs, one representing the resource allocation and statistics, the other the resource capacity. Interfaces are provided to access and modify the relationship among these graphs. The statistics on the graph are collected and updated by the State Estimator.

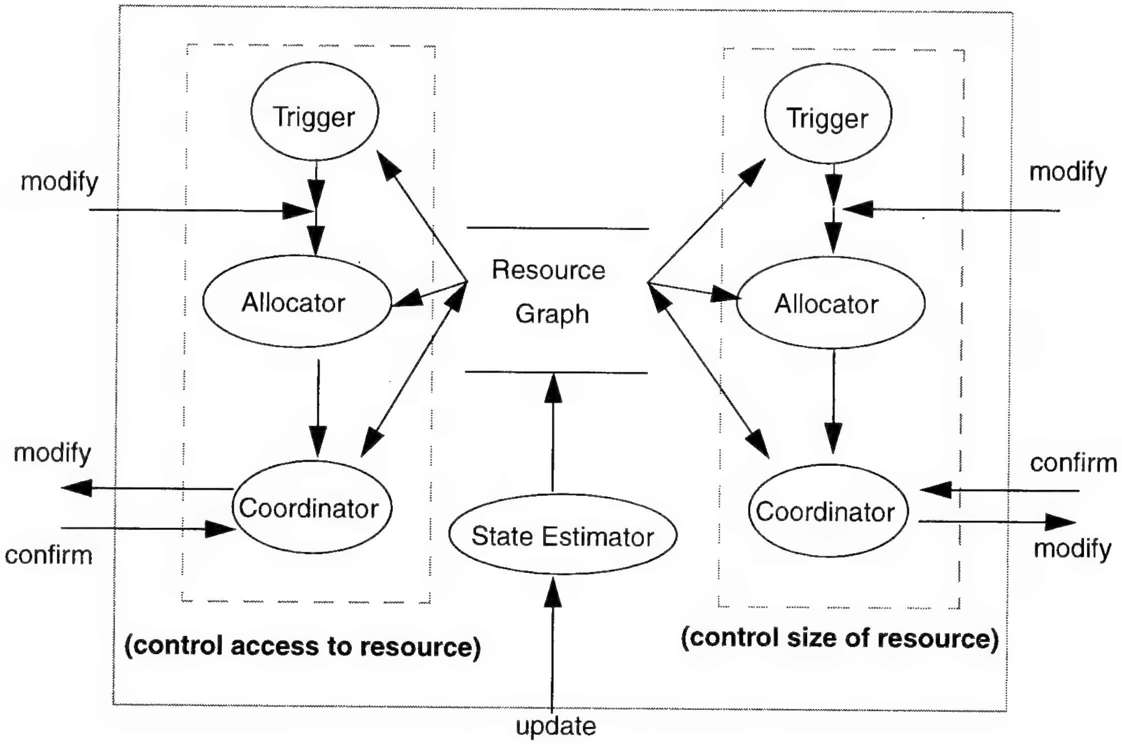


Figure 7 Functional model of a generic controller

In our implementation, a generic controller is realized as a container class in C++, which includes as base classes the subcontrollers, trigger, allocator, coordinator, etc. Interfaces offered by these subcontrollers are implemented as virtual functions that are overloaded for a specific realization of the controllers.

The design of the generic controller shown in Figure 6 has brought us the following benefits. First, it was possible for us to design and implement all three classes of controllers --VP controller, VPG controllers, and VPN controller-- as a refinement of the generic controller class. For example, the VP controller in Figure 6 has two "non-trivial" controller objects --the VC resource allocator and the coordinator-- and five "trivial" controller objects. (Trivial controller objects can be thought of as objects which perform no action except that of forwarding data to another object. They are not shown in Figure 6). The VPG controller contains four non-trivial controller objects and three trivial objects.

Second, based on the generic controller design, we were able to realize different control schemes that attempt to achieve different control objectives for the customer control system. Realizing different control schemes is often possible by exchanging a set of subcontrollers in the system. For example, we implemented two classes of VC capacity allocators, realizing different VP schemes. One scheme aims at achieving call blocking objectives related to performance classes. The other scheme realizes call preemption in case of congestion, taking into account the priority of a call. In the same way, we have realized different synchronization protocols by building different classes of coordinator objects. One of these protocols is designed towards efficient use of VPN bandwidth, another towards guaranteeing fair access to the VPN capacity in case of overload conditions.

2.6 Enabling Management Objectives

The customer operates a management system to control and monitor the traffic on the enterprise network. A part of this system manages the traffic over the VPN, which is the focus of this paper. Examples of management capabilities that are related to the VPN service include controlling the bandwidth cost of the VPN service, VP bandwidth management, and QOS management. In the following, we describe how the management objectives outlined in Section 3 can be realized.

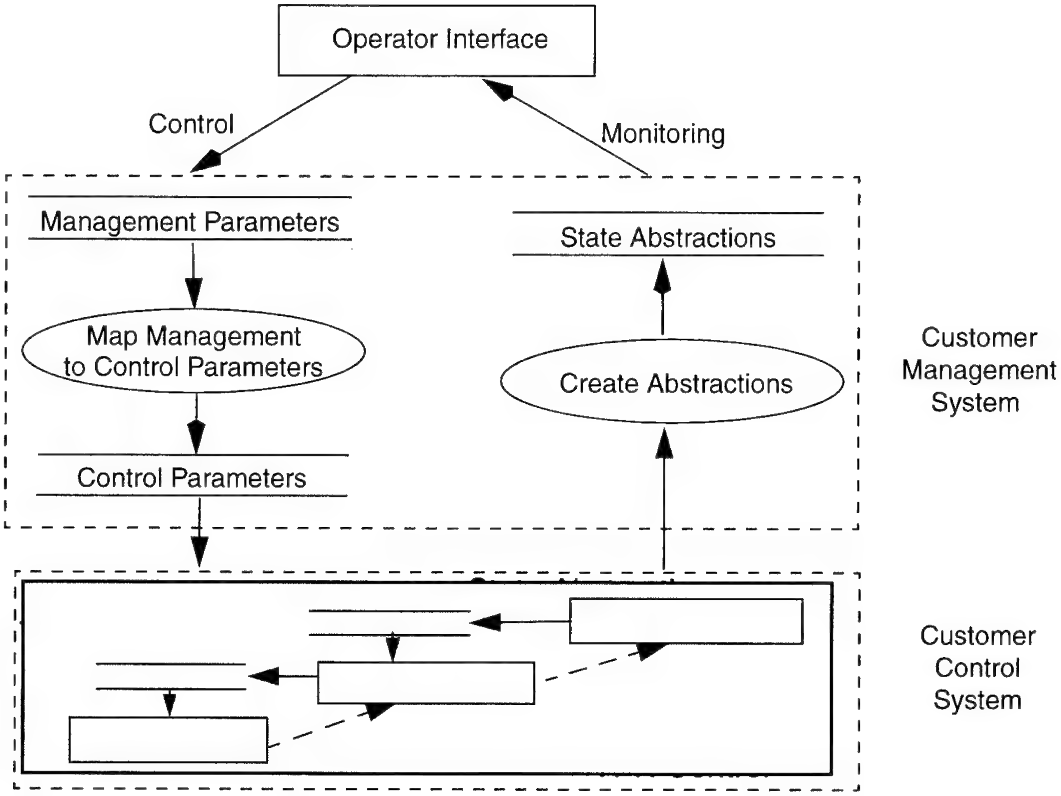


Figure 8 Framework for customer management

Figure 8 shows our framework for implementing management capabilities. In this framework, management parameters, which directly relate to management objectives, are mapped onto control parameters, which influence the behavior of the controllers, and are subsequently distributed to the controllers in the customer control system [PAC95]. In our implementation, management parameters are made available to the operator of the enterprise network through the management console (Figure 8).

A management parameter can be mapped onto control parameters for one or more classes of controllers. For example, cost management operations affect only the VPN controller. Allocating a specific capacity to a VP through a VP management operation affects both the VPG and the VPN controllers. QOS management operations, such as setting call blocking objectives, generally affect all classes of controllers. In response to a change in blocking objectives, the VP controller adjusts its VP policy, the VPG controller changes the VP allocation strategy, and the VPN controller negotiates the VPG sizes according to the new bandwidth requirements.

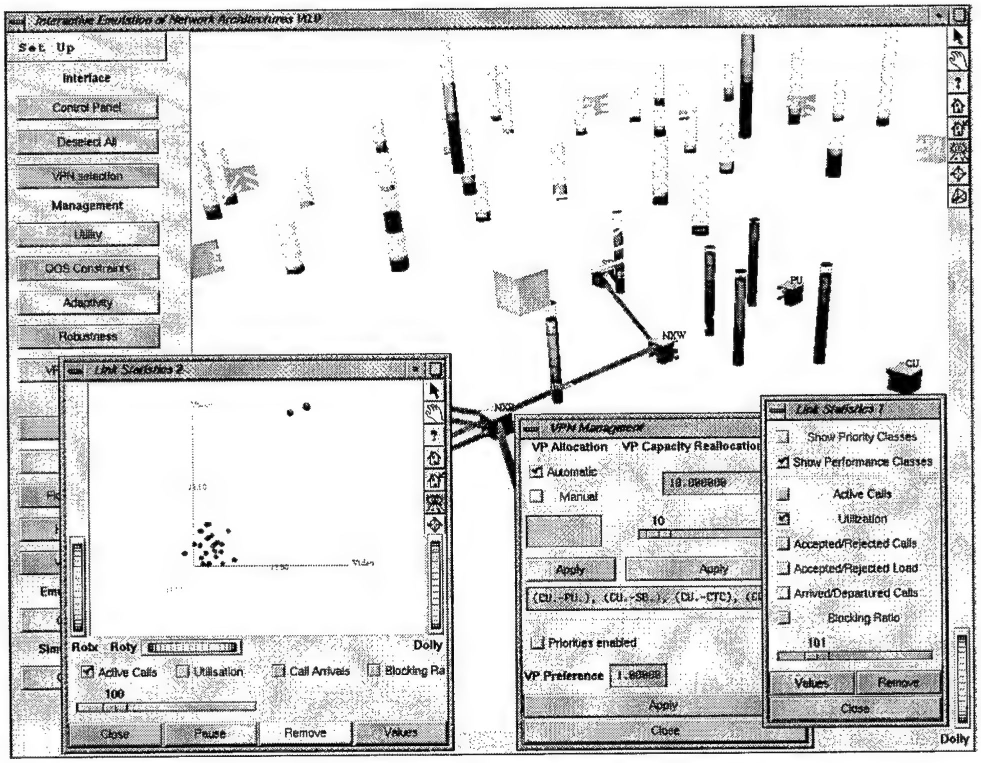


Figure 9 The customer management console for a VPG-based VPN service. The upper layer represents the VP network, the lower layer the VPG network. The vertical bars on the VP network indicate the utilization, the vertical bars on the VPG network the allocation of VPG bandwidth to VPs

Figure 9 shows the screen of the management console that we have implemented for customer management of a VPG-based VPN service. Both layers of the VPN are visible. The upper layer represents the VP network, the lower layer the VPG network. The vertical bars on the VP network show the current utilization of the VPs. The three segments of a particular bar correspond to the three traffic classes supported in our particular system. The outline of the cylinders indicate the currently allocated VP capacities. The vertical bars on the VPG network give the allocation of the VPG bandwidth to the VPs. A "cloud view" on the lower left corner shows the number of active calls in the VPs. Each axis corresponds to a traffic class. In this specific snapshot, one can see that two of the VPs experience a much higher load than the others. The interface in Figure 9 allows an operator to perform management operations and observe the reaction of these operations on the global state of the system.

3 QOS Guarantees and VPN Control Algorithms

3.1 Introduction

From a performance point of view, there are two goals:

- End-to-end QOS: The network must guarantee cell-level QOS end-to-end to all connections it admits. The QOS should, as far as possible, be independent of the characteristics of the underlying public network services.
- Efficient use of resources: The architecture must support statistical multiplexing of the customer traffic within the VPN, and it must allow dynamic bandwidth renegotiation on all control layers in order to accommodate changes in the traffic characteristics within the customer network.

The first goal is closely related to the problem of defining an abstraction of the VPN capacity for the customer network. (Here we understand the term capacity as a call-level abstraction, defining how many connections of a given class can be supported by the VPN, while observing the cell -level QOS requirements of each such connection.). VPN services may be provided by multiple providers and it is likely that each of these providers use different QOS guarantees. As a result, we argue that the requirements on the provider necessary to construct such a capacity abstraction should be minimum, for example, based on services that are already commonly available today. Our approach is presented in Section 3.2. The second goal is achieved by implementing a set of control algorithms according the control architecture outlined in Chapter 2. The algorithms implemented are described in Section 3.3 and Section 3.4. Finally, in Section 3.5, we present an evaluation of the performance of a system prototype implemented based on simulation.

3.2 Constructing a Call-Level Abstraction with Cell-Level QOS Guarantees

The network model we assumed is a multiclass network supporting a finite number of traffic classes like voice, video, and data, each of which is defined by two sets of parameters: the traffic characterization and the QOS requirements. The QOS requirements can be seen as constraints under which the real-time control system of the customer network must operate. They include parameters like the maximum cell delay, the maximum error rate, and the minimum average throughput per connection.

Different customers have different requirement on their networks, including defining their own real-time and non real-time traffic services (or classes) with specific cell-level QOS requirement. The customized QOS requirements are built upon services provided by multiple providers. Therefore, an approach is required to handle the different QOS schemes between the customer domain, and the VPN domains.

We approach the problem in two steps. First, we define a call-level capacity abstraction called the *Schedulable Region* [HYM93]. Use of scheduleable region allows the customers to define their own traffic classes according to their network requirements and provides a unified notion of resource capacity abstraction independent of cell-level algorithms and hardware specifics. Second, we describe how such an abstraction can be built using CBR VP services. We believe that the use of CBR VP is a reasonable approach because this service is one of the basic ATM services defined by the ATM Forum and is relatively easy to provide. In fact some public network providers have already begin to offer such services [ATS93].

3.2.1 Cell-Level QOS Guarantee through Schedulable Region

The task of the admission controller is to accept or reject calls so as to maximize some utility function under the constraint that the required QOS at the cell-level to every call admitted into service can be met. From the point of view of the admission controller, the link capacity can be expressed by its schedulable region S, which defines how many calls of a given class the link can support, while guaranteeing the appropriate cell-level QOS to each class. The cell-level QOS for a traffic class is specified in terms of bounds on loss probability and delay experienced by cells through a network multiplexer. The schedulable region S is an N-dimensional space, where N is the number of classes (such as voice, video, data) recognized by the link controller. The resource state is defined by the occupancy vector $\mathbf{x}$, which represents the number of calls of each class currently active in the link. In order to guarantee cell-level QOS to each class of traffic, the occupancy vector can assume only values that are inside the schedulable region. In general, the size of the region will depend on the statistical characteristics and cell-level QOS constraints of each class of traffic, as well as on the details of the scheduling policy in use.

For illustration purposes, we estimated the size of the scheduleable region for a 40 Mb/s link through simulation. In the simulation, we defined two traffic classes. The first traffic class (Class I) is for video. It is characterized by 24 frames per second, peak rate of 6Mb/s, average rate of 1.32 Mb/s and the cell-level QOS constraints are maximum delay of 1 ms through the multiplexer and no cell loss. The second traffic class (Class II) is defined for voice traffic and is modeled as on-off source with constant arrivals with an exponential distributed active period and 64Kb/s peak rate. The cell-level QOS requirements for class II are maximum delay of 1 ms and no more than 2% cell loss. A static priority scheduling is used. The schedulable region of a multiplexer with a 40Mb/s link is shown in Figure 10. The line in the figure delimits the schedulable region of the multiplexer.

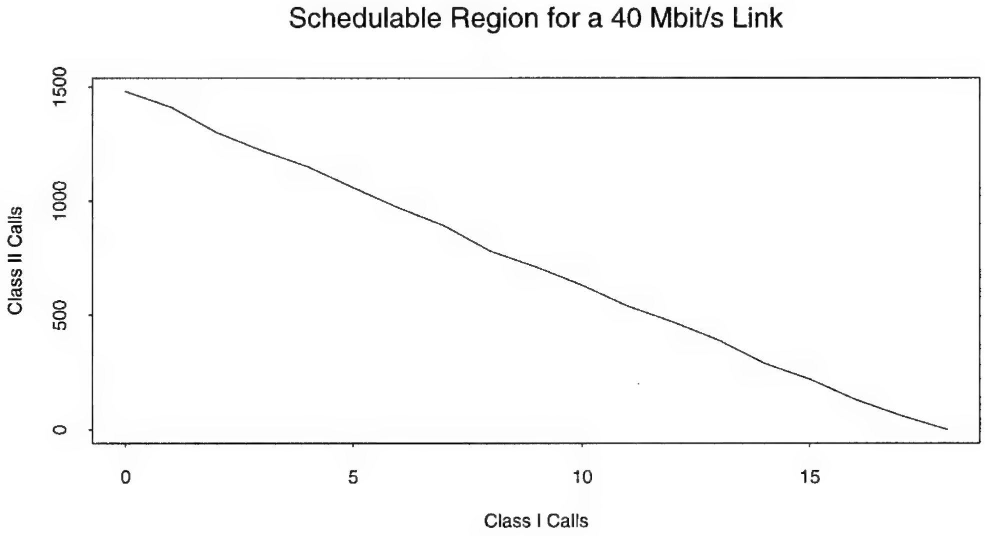


Figure 10 Schedulable region of a 40 Mb/s link

The admission controller must ensure that the number and class I and class II calls admitted onto the multiplexer is a combination below the boundary of the schedulable region. For an end-to-end connection, cell-level QOS guarantee is satisfied if at each multiplexer, the operating point (the combination of the class I and class II after the call is accepted) is below the schedulable region and if the following constraints are satisfied.

Delay Constraint

$$\sum_i s_{multiplexer_i} + \sum_j s_{link_j} < s$$

Cell Loss Constraint

$$\sum_i \varepsilon_{multiplexer_i} + \sum_j \varepsilon_{link_j} < \varepsilon$$

where s_i is the maximum cell delay and ε_ι is maximum cell loss at the multiplexer (or link) i.

The concept of a schedulable region allows a separation between cell and call control. As a result, the call admission control policy used in this framework operates only on call level abstractions. Cell-level QOS guarantee is enforced by restricting the occupancy state of the link to below the schedulable region. The admission control policy specifically deals only with call-level QOS like blocking probability. This approach is very different from many existing admission control algorithms where the admission controller operates on both call and cell statistics (see for example [LEE96]).

3.2.2 Constructing a Schedulable Region using CBR VP

In order for a scheme that uses schedulable regions to work, the traffic classes defined by the customer must be recognized by all of the multiplexing units. However, in the case of a Customer Network (CN), it is unlikely that this is true in the Virtual Private Network (VPN) domain, which is built on top of the public network. In the rest of the section, we will show how a homogenous view of cell-level QOS guarantee can be built even though the Customer Premise Network (CPN) cell-level QOS guarantee uses a cell multiplexing scheme different from the Public Network (PN) using Constant Bit Rate (CBR) Virtual Path (VP).

To discuss the abstraction of communication resources within the CPN domains, we refer to a general architecture of a broadband switch that is based on a non-blocking switch fabric (see Figure 11). Traffic arriving at an input link of the switch is routed onto one of the output links.

Note that the critical resources in the switch are the output ports, each of which is controlled by a link resource controller.

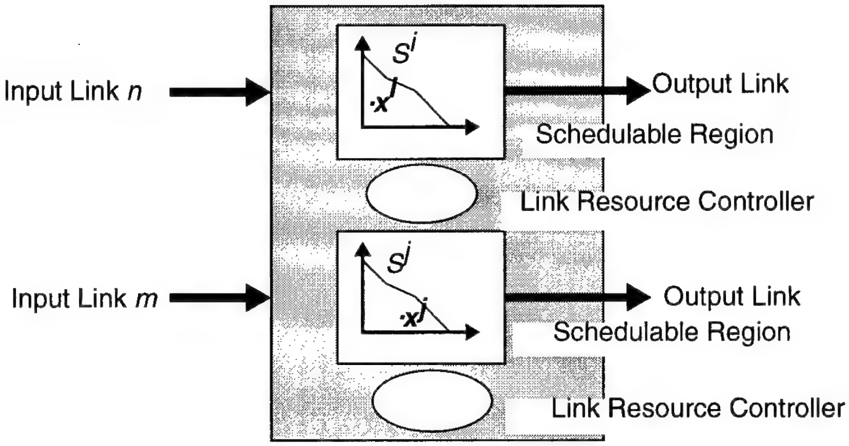


Figure 11 : Architecture of a typical broadband switching node in the customer network.

Given the cell-level QOS constraints, the size of the schedulable region is determined by the following:

- size of the output buffer
- the buffer management algorithm
- the capacity of the output link
- the scheduling algorithm
- the cell-level traffic statistics

We exploit the fact that both a physical link and a CBR VP are characterized by the same parameters, namely: bandwidth, cell delay and cell loss ratio. The difference is in the factors contributing to delay and cell loss. In both physical link and CBR VP, delay can be caused by propagation (which depends on the length of the cable) and transmission (which depends on the size of the packet and the speed of the link); cell loss on a physical link can be caused by errors in transmission or synchronization. For a CBR VP, delay can also be caused by switching and queuing, and cell loss through contention for buffer space in the public network domain. The following table summarizes the discussions

	Delay	**Error**
Physical Link	Propagation Delay (D_p^1) Transmission Delay (D_p^2)	Cell loss due to physical error (E_p^1)
CBR VP	Propagation Delay (D_l^1) Transmission Delay (D_l^2) Switching Delay (D_l^3) Queuing Delay (D_l^4)	Cell loss due to physical error (E_l^1) Cell loss due to buffer contention (E_l^2)

Table 2: QOS parameters for physical and logical links

Thus, given a CBR VP (of a specific bandwidth) with bound on maximum delay and loss, construction of schedulable region can proceed as if the output link is a physical link with the same bandwidth. The size of the schedulable region of the combination of the multiplexer and logical link of 45 Mb/s will be the same as if we have a 45 Mb/s physical link, with modifications needed to the cell-level QOS provided to take into account the additional delay and loss due to queuing, transmission and switching introduced by the CBR VP. In particular, the maximum cell delay is equal to the sum of the maximum delay over the multiplexer and the logical link. The maximum cell loss ratio is (1.0 - (1.0 - cell loss ratio over mulitplexer)*(1.0 - cell loss ratio over CBR VP)).

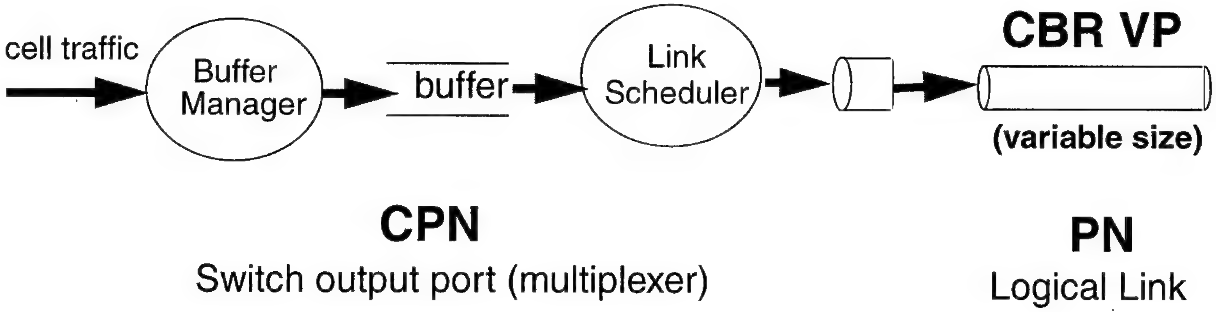


Figure 12 : Construction of scheduleable region using CBR VP

Using such an approach, the customers can choose traffic classes with QOS specifications that are totally independent of the provider. Negotiation of resource between CPN and PN is a matter of finding the logical link bandwidth corresponding to the desired schedulable region size.

As it is expected that bandwidth negotiation is done in discrete step size, the mapping between logical link size and schedulable region size can be computed off-line by the customer and put into a table. An example of such a table is given in Table 3 . The table has two columns. The right column contains possible bandwidth of the logical link, in this case measured in terms of megabits per second (Mb/s). The left column contains the corresponding size of the schedulable region for each logical link size. For example, if the size of the logical link is 15 Mb/s, the size of the schedulable region is *Region15*. By constructing such a table, dynamic bandwidth negotiation between the CPN and PN can be done in units of Mb/s, even though within the customer domain, bandwidth negotiation is performed in terms of schedulable region.

Logical Link Capacity (Mb/s)	**Schedulable Region of Multiplexer**
10	Region10
15	Region15
.....	.....
100	Region100

Table 3: Mapping between Link Bandwidth and Size of Schedulable Region

With the resource abstraction given in terms of schedulable region, control algorithms can now be designed so that they do not deal with the specifics of cell-level multiplexing and instead deal with only call-level capacity abstractions. This is the topic of the next section.

3.3 Review of VP Control Algorithms

Many classes of control algorithms can be implemented using the framework provided by the functional model and generic controller design described in Chapter 2. In particular, a large class of VP bandwidth allocation algorithms that was previously used mainly in the provider's domain can now be applied in the customer domain. Using the architecture framework described in Chapter 2, we characterized these VP algorithms according to the time-scale and the controller on which they run on.

A large class of VP allocation algorithms can be run by VPG controller(s). For a survey of some of these algorithms see [ANE96]. These algorithms tend to be centralized, and capacity distribution is computed based on medium- to long- term measured traffic statistics. The time-scale on which these algorithms run is very important to their design. Some of these algorithms are used by the network provider for dimensioning of the network size, which occurs on a very slow time-scale. On the other hand, in [LOG95] a "medium time-scale" VP redistribution algorithm is proposed. In this paper, the author highlights the importance of choosing the correct redistribution period. It is argued that the bandwidth reallocation time (BRT) should be large enough so that existing traffic has the time to leave the network. For traffic with exponential holding time of mean 100s, they suggest a BRT of 30 minutes (1800s).

VP bandwidth reallocation algorithms can also be run by a VP controller. These algorithms are usually triggered by specific changes in the VP utilization. For example, [OHT92] and [ORD96] describe two state-based dynamic VP bandwidth renegotiation algorithms. In [OHT92], a fixed amount of bandwidth is requested when the utilization of the VP crosses a high "water-mark". On the other hand, bandwidth is released when the utilization crosses a low "water-mark". A similar approach is used in [ORD96]. The difference (and improvement) comes from requesting and releasing variable amounts of bandwidth, which will depend on the call arrival and departure statistics. The algorithm described in [ORD96] is more efficient but is significantly more complex to implement than the algorithm in [OHT92]. The time-scale on which these algorithms run depends on the call arrival and departure statistics.

Control across different control layers can be coordinated in order to achieve a possibly better system performance by organizing the system as a hierarchical structure. This is the approach taken in [PIT95], where multilevel optimal control is performed. A three-level hierarchically organized control structure is used. The overall objective is to optimize objective functions that minimize the sum of the difference between the desired and measured delay and switching bandwidth allocated to a specific traffic class within a VP. On the lowest level, the local controller solves a set of low order linear difference equations that assumes that the predictions on the queue state and allocated service rate are exact. On the higher level, the controllers attempt to force the prediction of the delayed terms equal to their true values. Successive iterations (if they converge) will ensure that the predicted trajectories are the same as the true trajectories. It should not be difficult to see that this control structure can be implemented in a rather straightforward way using our architectural framework.

3.4 Realization of the Control System

In this section, we provide a detailed description of an implementation of the control system architecture described in Chapter 2. In particular, we would like to highlight new possibilities available due to the introduction of the VPG concept. Specifically, since VP bandwidth allocation can be performed entirely within the customer domain without interacting with the provider, we implemented a centralized VP algorithm that exploits multiplexing on the VPGs among VPs from different source-destination pairs. This algorithm runs on the VPG controller and periodically redistributes the VP bandwidth based on the current state of the VPs, and the measured call arrival and departure statistics. The bandwidth of all VPs is changed simultaneously, and needs to be performed frequently for efficient distribution of VP capacities. In a VP-based VPN, such an algorithm will be considered impractical due to the frequent interactions it needs to make with the public network provider management system. It is, however, a reasonable choice in a VPG-based VPN, since all control interactions are performed within the customer domain.

The algorithm we have developed is different from those described in [OHT92] and [ORD96] in the following way. First, although redistribution is based on the current state of VPs, the algorithm is centralized and runs in the VPG controller, instead of being distributed and running on the VP controller. Second, redistribution is performed periodically, every T seconds, instead of being triggered by a change in the state of the VP. The parameter T provides an explicit and direct way in which the management system can influence the behavior of the VPG controller.

3.4.1 VP Controller

The admission control algorithm implemented on the VP controller is based on the Multidimensional Threshold Control policy (MDT), a form of complete partitioning (CP) policy [HYM91]. This policy is designed to achieve the blocking constraints for each class by the imposition of thresholding rules. Let z be the threshold vector, S be the schedulable region and, x = (x^1,...,x^n), where x^i is the number of calls in the system of class i; then define the policy u as

$$u^i(x) = \begin{cases} 1 \; if(\{x^i < z^i\} \wedge \{x + e^i \in S\}) \\ 0 \qquad\qquad otherwise \end{cases}$$

where $u^i(x) = 1$ if a call of class i can be accepted, and 0 if otherwise.

In this policy, each class, in effect, is allocated its own dedicated bandwidth, and no call-level interference between classes takes place. Modeling call arrivals as Poisson processes and assuming that the holding time of calls is exponentially distributed, the blocking probability p^i for class i is given exactly by $p^i(u) = E(\lambda^i / \mu^i, z^i)$ where E(A,N) is the one-dimensional Erlang-B formula:

$$E(A, N) = \frac{A^N / N!}{\sum_{i=0}^{N} A^i / i!}$$

where κ^i is the blocking objective for class i. The threshold z^i is thus chosen as the smallest (or minimum) amount of bandwidth N such that the blocking probability calculated using the one-dimensional Erlang-B formula is less than or equal to the blocking constraint κ^i. Mathematically, this is expressed as the following equation:

$$z^i = \min_{E(\lambda^i / \mu^i, N) \le \kappa^i} N$$

The CP admission control algorithm used has the advantage that it tends to divide the resources "fairly" among the traffic classes according to the blocking objectives. By computing the thresholds used in the policy dynamically during run-time, the algorithm is also able to adjust to variations in traffic load. Perhaps, more importantly, the behavior of this algorithm can be influenced by management operations by changing the values of κ^i, the blocking objective for class i traffic. Changing κ^i causes the corresponding threshold z^i used by the MDT algorithm, and thus the admission policy, to be changed.

Conceptually, the surface of the schedulable region can be highly irregular. However, the storage and manipulation of a general N-dimensional space can be rather expensive. Therefore, all surfaces are approximated as a N-dimensional hyperplane, which can be expressed in the form $\sum x^i / N^i = 1$, where N^i is the maximum number of calls of traffic class i that can be allowed into service. Algebraically, the approximation involves finding all the N^i. There is, however, no unique solution to the approximation. As shown in Figure 13., more than one possible approximation is possible. (In fact, there are infinitely many of them). One way to decide on an approximation is to fix the ratio N^i/N^j, for all i not equal to j. Some possibilities for these ratios are to let N^i equal to the average bandwidth, peak bandwidth, utility or some combinations. Another approach is to choose the surface that maximize the utility function $\sum u^i N^i$, where u^i is the utility of class i calls. Figure 13 shows two possible approximations for a 2-dimensional schedulable region.

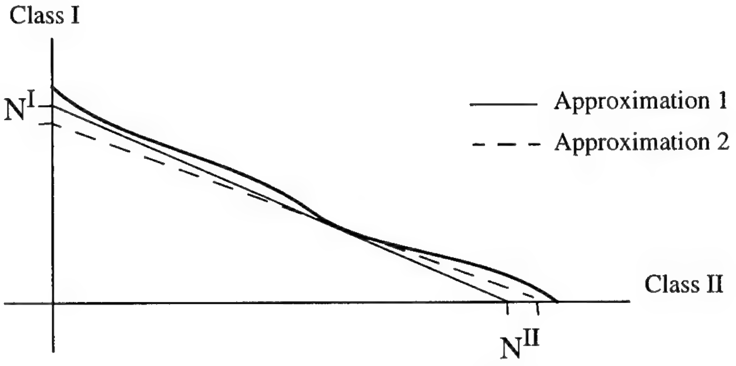


Figure 13 Approximating the scheduleable region with a hyperplane.

Depending on the size of the scheduleable region, it is possible that the hypercube formed by the boundaries of all the thresholds will be either under or above the boundary of the schedulable region. In the case where the hypercube is below the boundary of S, the thresholds are increased to allow for better utilization of the resources; in the case where the hypercube is above the boundary of S, the thresholds are decreased so as to maintain some sort of "preference" among traffic classes. The heuristic in this case is to use a new set of thresholds (z^i_p) that forms the largest hypercube that is below the boundary of S and such that the ratios among thresholds are maintained. Since the schedulable region is represented as a hyperplane, the new thresholds can be computed using a simple vector projection operation given as follow:

$$F = \frac{1}{\sum \frac{z^i}{N^i}} \qquad z^i_p = F \times z^i$$

Figure 14 illustrates the projection operation F graphically. In Figure 14(a), the thresholds calculated form a rectangle below the boundary. In order to increase utilization, the thresholds are increased by the projection operation into a larger box. Figure 14(b) shows the reverse case where the rectangle forming the computed threshold is above the boundary. The thresholds are decreased using the projection operation so that the new thresholds are smaller and the rectangle is below the boundary surface.

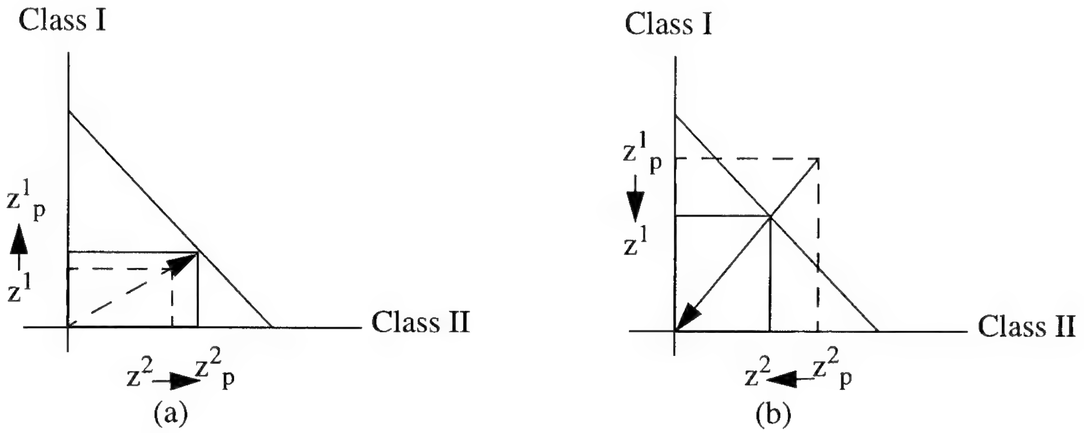


Figure 14 : Projection of thresholds

During durations of overload, when the demand for bandwidth is much higher than the amount of bandwidth available, the control objective could be different from control under normal conditions. In particular, during periods of overload, it might be more important to guarantee the blocking constraints of certain classes of traffic than for others. Such control objectives can be achieved through the use of a priority scheme.

A straightforward implementation of a priority scheme is to accept a call of higher priority as long as there is sufficient capacity available. Pre-emption of calls of lower priority is allowed if necessary. Such an implementation poses a problem in terms of blocking objectives in the sense that calls of high priority do not experience blocking at all, as long as there are calls of lower priority in the system. The blocking probability of calls in a certain traffic class then depends only on the capacity of the VP, and has nothing to do with the blocking constraint of the class.

One possible solution is to define two separate objectives. One objective deals only with blocking constraints and the other deals only with priority. Admission control can thus be executed with either one of these objectives. Blocking constraints can be used for example in normal load conditions, and priority used in overload conditions.

The approach we took is to integrate both objectives into the same framework. Each call is associated with a performance characterization and a priority. Priority is considered purely as a policy (when there is no spare capacity, high priority pre-empts low priority calls). Blocking remains then as the only objective to be met, and the priority of a call implies how faithfully (or with what priority) its blocking objective should be respected. Once the actual admission control algorithm implemented, a variation of the MDT algorithm enhanced with the notion of priority, is described as followed:

(1) When a call with blocking objective K^i and priority L^i arrives, if the link utilization is less than α (< 1.0), accept the call or else go to (2).

(2) Accept the call with probability $(1-K^i)$ and reject the call with probability K^i. If the call is to be accepted and the VP is full, pre-empt the minimum number of calls with lower priority such that this call can be accepted. If there are not enough calls of lower priority in the system such that this call can be accepted, the call is blocked.

The algorithm implemented has the following characteristics. First, in times of overload, calls of higher priority will experience actual blocking probabilities close to the specified objectives (as long as there are calls of lower priority to pre-empt). Second, when the value α is set to a value less than 1.0 and the system is not heavily utilized, calls of higher priority can take advantage of the availability of resources and will experience blocking probabilities less than the specified objective. This algorithm has the advantage that it works with any level of priorities. Experience from the simulation shows that the algorithm is simple to implement and can be executed very efficiently.

3.4.2 VPG Controller

In order to exploit statistical multiplexing among VPs from different source-destination pairs, the VPG controller periodically redistributes the capacities among the VPs. The algorithm takes into account the blocking objectives, the period of distribution, the current operating point of the VPs, and the call arrival and departure statistics measured.

The implemented centralized algorithm runs periodically. The algorithm executed in each cycle is described as follow:

Step (0): Check if the previous computation has terminated. If it has not, go to (5)

Step (1): For each VPG i, calculate the weight of VP j on VPG i. Let this weight be W_i^j. Based on this weight, calculate the bandwidth that VP j (BW_i^j) is getting from VPG i, whose bandwidth is BW_i using the equation

$$BW_i^j = \frac{W_i^j}{\sum_j W_i^j} \times BW_i$$

Step (2): For each VP j, calculate the amount of bandwidth it is allocated, BW^j, using the equation:

$$BW^j = min(BW_i^j)$$

Step (3): If there is any unassigned bandwidth left in any VPG, form a candidate list containing all the VPs. Pick a VP at random and assign it the maximum possible bandwidth. Remove this VP from the list. This step ends when either all VPG bandwidth has been assigned or there are no more VPs left in the list.

Step (4) (a): Divide the VPs into two groups: “small” and “large”. VPs whose new allocations are less than the current allocation are in the “small” group, or else they are in the “large” group.

Step (4) (b): Send out an allocation of VPs in the “small” group and wait for acknowledgements from all these VPs.

Step (4) (c): All acknowledgments have been received. If the allocation fails on any of the VPs, the allocation fails. The original VP bandwidth is sent out to all VPs in the “small” group to execute a rollback operation. If all allocation on “small” VP succeeds, send bandwidth to “large” VPs.

Step (5): End of computation cycle.

Step (4) is necessary due to the fact that the states of the VPs are distributed. Changing these states has to be coordinated so that at any time, the sum of the capacities allocated to the VPs do not exceed the capacity of the VPGs they pass through. The solution is to divide VPs into two groups. The first group ("small" group) consists of VPs whose new allocations are smaller than the current allocation; the second group ("large" group) consists of VPs whose new allocations are larger or equal to the current allocations. In the first phase, only allocations of VPs in the "small" group are sent, in the second phase allocations of VPs in the "large" group are sent. Note that VP controllers who receive allocations in the second phase have larger or equal allocation and therefore have no problem.

When the VP controller receives a request for a change in bandwidth and cannot execute the request immediately, it can react in two ways. It can either reply immediately with an unsuccessful message or it can stop accepting new calls, while allowing existing calls to depart. Eventually, its utilization will go below the new allocation, and when this happens, the VP controller replies with a successful message and can again accept new calls. The first solution has the advantage that the duration of an allocation cycle is bounded and usually short. It can, however, fail to change the VP bandwidth frequently when the system is heavily loaded. As a result, VPs that experience a heavy load earlier tend to have more bandwidth allocated. On the other hand, the duration of an allocation cycle in the second solution can be long. However, even with a heavy load, the allocation always succeeds.

Using a reasonable value for the period of computation, a computation cycle in the first solution usually terminates before the start of the next cycle. If the previous computation has not been completed, a new allocation is not allowed to start so that only one allocation process is allowed at a time (Step 0).

If the VPG controller receives one or more unsuccessful messages, it aborts all changes performed in the first phase. This operation is similar to the roll-back operation in a 2-phase commit protocol used in database management system. The second phase will not be executed.

Note that it is not specified in step (1) how the weight of a VP should be computed. In the following, we present two possible approaches to computing the weight of a VP. The first approach is computationally intensive and is not suitable for real-time control. It can, however, be considered as the reference case for the second approach, which is based on a heuristic that attempts to approximate the first approach.

3.4.3 Exact Approach to Weight Computation

We formulate the exact approach as follow.

For a given traffic class *x*, we model the VP as a M/M/C/C queue where C is the maximum number of calls of class *x* it allows into service. The dynamic behavior of the states of the VP is described by the Chapman-Kolmogorov equation for continuous-time Markov chains [KLE75]:

Let $P_{ij}(t)$ be the probability that the state of the VP goes from initial state i to state j in time t. The set of differential equations that describe the transition from state i to state j in time t are:

$$\frac{dP_{i0}(t)}{dt} = -\lambda P_{i0}(t) + \mu P_{i1}(t)$$

$$\frac{dP_{ij}(t)}{dt} = \lambda P_{i,\,j-1}(t) - (\lambda + \mu j)P_{ij}(t) + \mu(j+1)P_{i,\,j+1}(t)$$

$$\frac{dP_{iC}(t)}{dt} = \lambda P_{i,\,C-1}(t) - \mu C P_{iC}(t)$$

$$P_{i,\,C+n}(t) = 0, n > 0$$

where λ = arrival rate of class *x* calls and μ = service rate of class *x* calls. These equations can be rewritten in matrix form as:

$$\frac{dP(t)}{dt} = AP(t)$$

where P is the matrix whose element at the i row and j column is $P_{ij}(t)$. Solving these differential equations translates into finding the eigenvalues and eigenvectors of a CxC tri-diagonal matrix. The solution is given in [RIO62] as:

$$P_{ij}(t) = \frac{\rho^j/j!}{\sum \rho^j/j!} + \frac{c!}{j!}\rho^{C-i}\sum_r \frac{D_i(r)D_j(r)}{rD_C(r)D'_C(r+1)}e^{r\mu t}$$

where $\rho = \lambda/\mu$, r is an eigenvalue of A and $D_i(r)$ is defined recursively as:

$$D_0(r) = 1$$

$$D_1(r) = \rho + r$$

$$D_{n+1}(r) = (\rho + n + r)D_n(r) - \rho n D_{n-1}(r)$$

The eigenvalues of A are real and simple (pp. 82, [RIO62]). The solution to $P_{ij}(t)$ nicely captures the dynamics of the transitional behavior of the multi-server queue. The first term of the equation is the equilibrium probability p_j, forming a truncated Poisson distribution. The largest negative eigenvalue, say r_1, governs the rate of approach to the steady state. For ρ much larger than C, this root is near $-\rho$; for ρ=C, it is -2; it approaches -1 as ρ approaches 0.

Let K^x be the blocking constraints associated with each traffic class *x*. Given that the state of the VP at time t is i, and the period of bandwidth recomputation is T, the smallest amount of bandwidth needed to satisfy the required blocking constraints is the smallest C that satisfies the following equation:

$$\frac{1}{T}\int_t^{t+T} P_{iC}(t)dt \le K^x$$

The value of C for each traffic class is computed separately. The weight of a VP is the bandwidth that corresponds to the smallest schedulable region that contains the hypercube formed by these thresholds.

Note that by using these weights, if sufficient capacity is available in the network, the VP controllers will always be able to guarantee the cell- and call- level QOS guarantee.

In order to show that the blocking probability can be calculated this way, we need to show that the PASTA property holds in a finite time. This is in fact true. The proof can be found on pp. 295 of [WOL89] and will not be reproduced here.

On a IBM/390 series workstation, using the LAPACK software[AND94], the computation time for finding the eigenvalues and eigenvectors of a CxC matrix are as follow:

Dimension (C)	Time (Seconds)
100 x 100	0.26
250 x 250	2.44
500 x 500	17.8
1000 x 1000	226

Table 4: Performance of LAPACK on a IBM/390

Numerical instability, in the form of complex and non-simple eigenvalues, is observed when the ratio C/ρ is too big. For example, if $\rho = 30$, all eigenvalues computed are real and simple only if C is less than or equal to 113.

Based on the actual eigenvalues computed, it is further observed that all eigenvalues must be used in the computation of $P_{ij}(t)$, even if these eigenvalues are very small (which should make $e^{r\mu t}$ very small). This is because the coefficient of the terms can be very large, and the final product is not negligible.

This exact approach to weight computation is obviously not practical for a real-time implementation, but it serves as a good reference for judging the performance of the heuristic algorithm to be presented next.

3.4.4 Heuristic Approach to Weight Computation

The heuristic weight computation is an attempt to approximate the weight computed in the exact case and is given by the equation below:

$$Weight = \sum_{x} \frac{-\log K^{x} + (\rho \times T \times 0.01) + i}{N^{x}}$$

The heuristic is based on the assumption that when the queue is empty, a certain minimum amount of bandwidth is needed to handle the anticipated arrival within the next computation period. The additional amount of bandwidth required increases linearly with the number of calls in the system. For each traffic class, there are two components to the weight. The first is the initial offset, which depends on the blocking objective, the traffic intensity and the computation period. The second component varies linearly with the current number of calls. The denominator is the normalization constant. A comparison with the exact values obtained in Chapter 3.4.3 shows that this approximation is reasonably close to the exact values in the range of T between 1s and 100s.

3.5 Evaluation of the Control System

The performance of the implementation described in Section 3.4 was evaluated in a scenario based on the topology of the NYNET testbed. Only the VPG and VP control layers were implemented. An important characteristics of our evaluation is that we run an executable model of the control system that runs on top of the emulation platform described in chapter 5. Each controller is implemented as a C++ object, and we simulate the exchange of signalling messages. Both delays and message passing time are modeled in the simulation.

In this scenario, a VPN service interconnects 6 CPNs. The VPN contains 14 unidirectional VPGs which support 30 unidirectional VPs, connecting the 6 CPNs in a full mesh topology. The two VPGs in the middle carry 9 VPs, the remaining VPGs carry 5 VPs each (Figure 15).

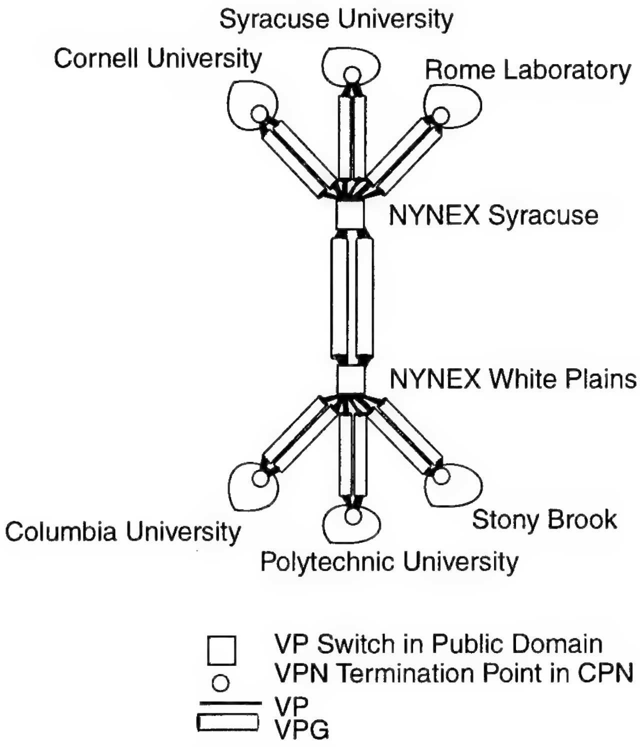


Figure 15 Network topology used in the evaluation

We model the network traffic load, the processing time of controllers and the time delay to send a message from one controller to another. For the sake of simplicity, all message processing time and message delivery time is set to 1ms. The computation time for a new VP distribution is assumed to be 100ms in all cases.

The network traffic is composed of two classes with different bandwidth requirements. A class 1 call needs ten units of bandwidth, while a class 2 call requires 1 unit of bandwidth. The holding time of the calls of both classes is exponentially distributed with a mean of 100 seconds, and call arrivals are modeled as Poisson processes. The arrival rate for class 1 call is 0.12 call/second and class 2 call 0.50 call/second. Finally, the blocking objectives are 0.10 for class 1 calls and 0.01 for class 2 calls. All VPs in the VPG link experienced the same offered load. The VPG links between Syracuse and White Plains have 180 units of bandwidth, and all other VPG links have 100 units of bandwidth.

We vary the bandwidth computation period (T) in the experiment and measured network blocking rate for three cases:

- Static VP bandwidth allocation. Each VP is given 20 units of bandwidth and the MDT algorithm is used for admission control.
- Dynamic bandwidth VP bandwidth reallocation using exact weight computation and the MDT algorithm is used for admission control.
- Dynamic bandwidth VP bandwidth reallocation using heuristic weight computation. A complete sharing policy is used in the admission control algorithm.

Figure 16 and Figure 17 show the blocking rates for both Class 1 and Class 2 calls. From the figures, it can be seen that both the exact and heuristic algorithms perform significantly better than static allocation and the blocking objectives are met by these dynamic algorithms in most cases. As expected, the performance of the exact algorithm deteriorates as the computation period increases. In fact, when T is large enough, the exact algorithm becomes the static allocation algorithm. As a result, the performance of the heuristic algorithm can be better than the exact algorithm when T is large. From Figure 16, we can see that for T > 30s, the heuristic algorithm outperforms the exact algorithm for class 1 calls. On the other hand, it is also true that if it is possible to use a T that is small enough, the blocking rate of a system running the exact algorithm can be made very small. This is not true in the case of the heuristic algorithm.

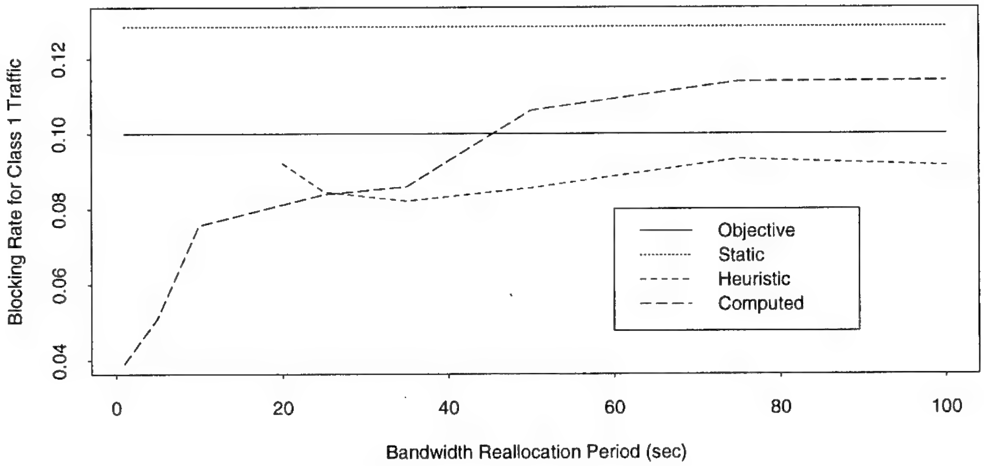


Figure 16 : **Performance of VPG Algorithms for Class 1 Traffic**

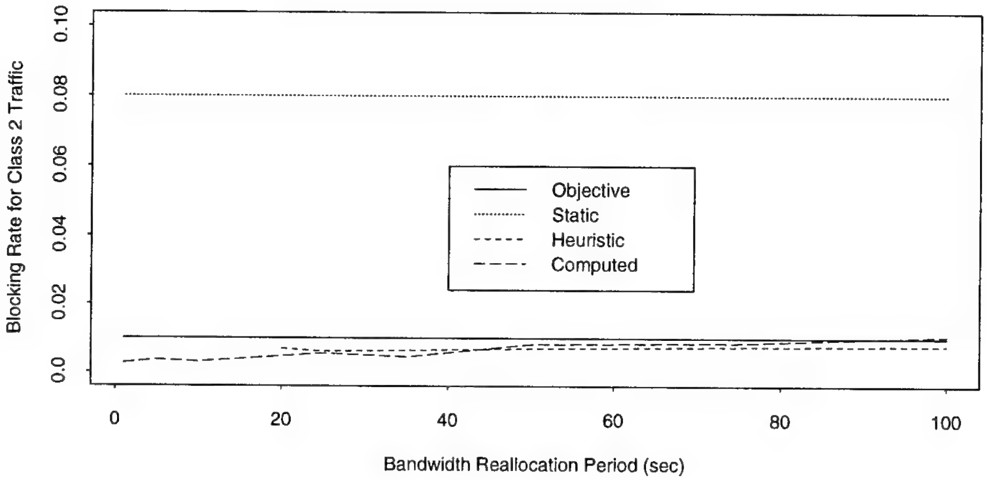


Figure 17 : **Performance of VPG Algorithms for Class 2 Traffic**

4 The High Performance Platform for Experimentation

4.1 The Platform

The platform used for implementing the prototyping environment consists of a supercomputer (either the KSR-1 or SP2) located at the Cornell Theory Center (CTC) in Ithaca, New York, connected to an SGI Indigo2 at Columbia University in New York, New York. These two machines are connected by a permanent virtual circuit connection (PVC) through NYNET, an ATM network that connects several research laboratories in New York State.

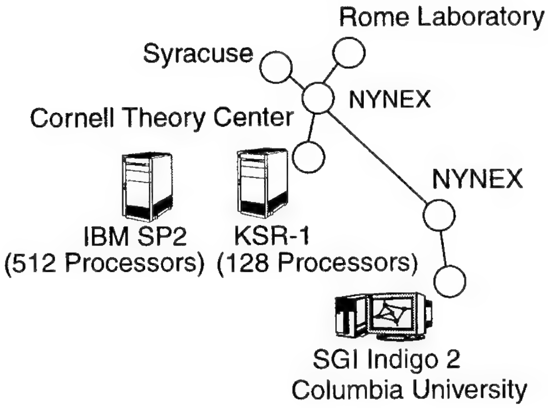


Figure 18 Hardware configuration of the interactive emulation platform.

The bandwidth B required by the connection is given by B = LM * NO * R, where LM is the size of the monitoring data unit, NO is the number of objects being monitored, and R is the refresh frequency. When NO=500 objects, LM = 512 bytes, R=1/sec, B is approximately 2 Mbit/sec. When the communication between the two machines is performed over Internet, both the number of objects monitored and the refresh frequency must be reduced to accommodate the lower bandwidth connection. In this case, our experience shows that, even with a reduced monitoring load, the packets carried over the Internet connection can experience delay variations of up to a few seconds. The ATM connection currently in use has been shown to be very useful for our purposes, providing higher throughput with lower delay jitter.

The parallel simulation kernel has been implemented on two platforms with different hardware architectures: the IBM SP2 (distributed memory) and the KSR-1 (shared memory). The simulation kernel was first implemented on the KSR-1. These implementations differ, in particular with respect to realizing message passing and referencing of objects. For the point of implementing a simulation kernel, message passing is important because the programming model of simulation is one of objects (or logical processes) exchanging messages. Therefore, while message passing is the “natural” programming paradigm on the SP2, we have to “imitate” message passing on the KSR-1 using shared memory.

In the remainder of this chapter, we will describe some of the knowledge we have gained through using these supercomputers, in particular the performance of simulating message passing on the KSR-1 and the porting process.

4.2 Building an Emulation Platform on the KSR-1

The KSR-1 is a shared memory parallel machine with hardware support for maintaining cache consistency. Each node has a 40-MFlops processor and 32 Mbytes of local cache memory. Processors are organized in ring hierarchy. Ring 0 has 32 processors and ring 1 has 34 ring 0s (Figure 19). The memory access time follows the same hierarchy. Memory access takes less than 1ms within the local processor, 8.75ms within the same ring 0, and 30ms within the same ring 1. The operating system is MACH, and the OSF-1 pthread package is provided to exploit the shared memory architecture of the machine.

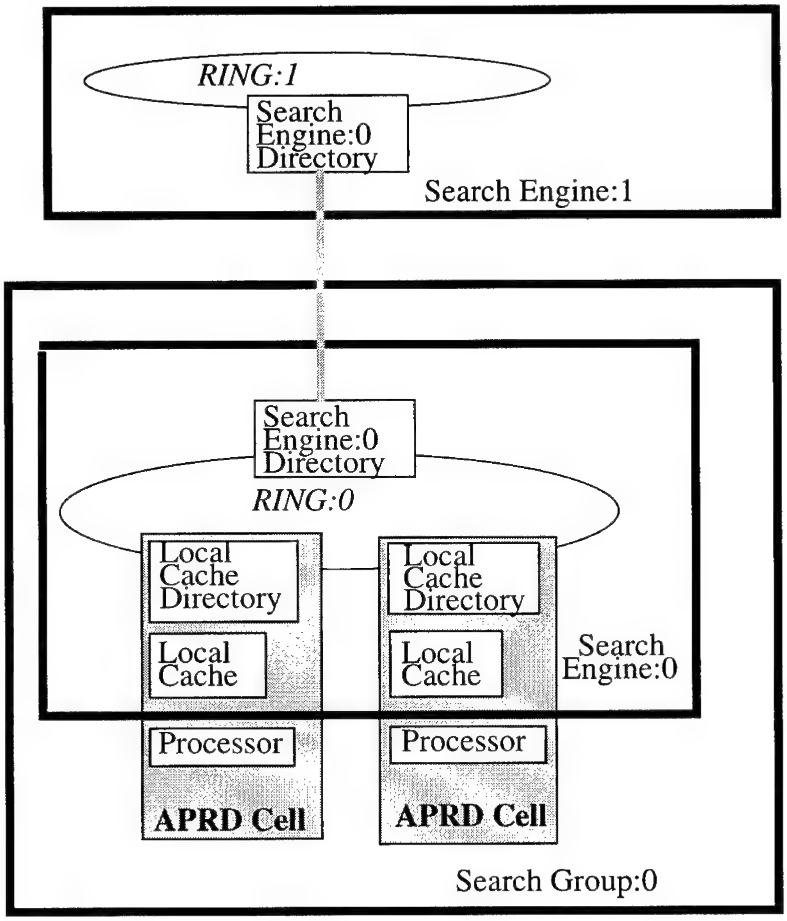


Figure 19 Hardware configuration of KSR-1

4.2.1 Benchmark Measurement on the KSR-1

As the KSR-1 was our initial experimental platform, we spent substantial time on the study of emulation performance of the KSR-1. The primary goal of this study was to estimate the performance of emulating large networks on a KSR, to identify major performance bottlenecks in an emulation system, and to suggest a system structure for emulation that reduces the impact of these bottlenecks. We emulated network mechanisms as (operating system) threads and implemented a message passing system to support asynchronous communication among mechanisms. Emulating large broadband networks thus involves executing a large number (hundreds) of threads of control, which cooperate in an asynchronous manner, using a message passing scheme for inter-thread communication. Operating system threads on the KSR-1 are available to the programmer as *pthread*, a POSIX threads standard (P1003.4a).

We did two kinds of performance experiments. First, we measured the performance of implementations of the bounded buffer paradigm [AND87]. In the bounded buffer problem (also known as the producer-consumer problem), two processes shared a common, fixed (and bounded) size buffer. One of them, the producer, puts information into the buffer, and the other one, the consumer, takes it out. Potential contention arises when both the consumer and producer want to access the buffer at the same time, and when the buffer is either full or empty. We chose to study this problem because of the way it captures the programming model of the message passing scheme used. Secondly, we studied the performance of a traffic control system that runs on top of our message passing scheme.

In our implementation of the bounded buffer paradigm, we found that the numbers of producers and consumers, as well as the ratio of consumers/producers, are the parameters that show the most significant impact on the performance. Having one producer and one consumer per buffer gives the best performance. Increasing the number of producer/consumer pairs results in a sharp drop in performance, roughly with O(1/n), with system size n. Also, the farther the ratio producer/consumer is from 1, the worse the performance gets.

We also found that synchronizing multiple pthreads, using mutual exclusion locks (or mutex locks) and condition variables [BOY93], gets prohibitively expensive, when the number of threads waiting on the same condition variable increases. We therefore conclude that the number of threads that access the same critical region should be kept to a minimum, in order to prevent a drastic decline in system performance.

In general, the performance depends on the pattern of interaction among nodes of the system. Take the example of two systems with extremely different communication patterns: a point-to-point communication pattern and a point-to-multi point communication pattern. If we express the performance of these systems as the number of messages per second to be exchanged between nodes, we get the following peak performance on a 64 processor set: 980,000 messages/sec for the former case and approximately 160,000 messages/sec for the latter. As mentioned above, these systems exhibit extremely different communication patterns. For a typical subsystem of a broadband network, we expect the communication pattern to be a mixture--broadcasts among subsets of nodes as well as one-to-one interaction--and, therefore, we estimate its peak performance to be in between the above given figures.

4.2.2 Experiences with Emulation of a Traffic Control System

By emulating a simple traffic control system (TCS), one that involves performing resource allocation tasks and executes a signalling protocol among network mechanisms, we encountered three problems that made stable performance for emulating a large network difficult: performance degradation due to serialization of KSR system calls, synchronization among mechanisms, and extensive context switching caused by the fine grain application we ran. We briefly summarize the problems and solutions we implemented in the emulation system.

Serialization of KSR System Calls

Some system calls in the KSR operating systems are serialized: “malloc()” and “free()” are two of them. For systems that perform many memory allocation calls, mechanisms spend an increasing amount of time waiting to acquire the lock that protects the global data structure, as the system size (and the number of mechanisms) gets large.

We verified this hypothesis through two measurements. We ran our traffic control system with two different libraries. The first library uses “malloc()” and “free()” with no consideration for their cost due to serialization. The second library uses an application-level memory management scheme that attempts to minimize these calls. In order to observe the effect of serialization, only mechanisms within the same network node interact. This is done by modifying the routing policy such that it generates only the address of the link controller in the same node. Since interacting mechanisms run on the same processor, the impact of remote memory access, contention, and thread scheduling is minimized.

Figure 20 shows the difference in performance between using and not using application-level memory management. When application-level memory management is not used, the throughput is low for a small system size and remains at that level as the system size increases. With application-level memory management, the throughput is significantly higher for systems of up of 128 network nodes.

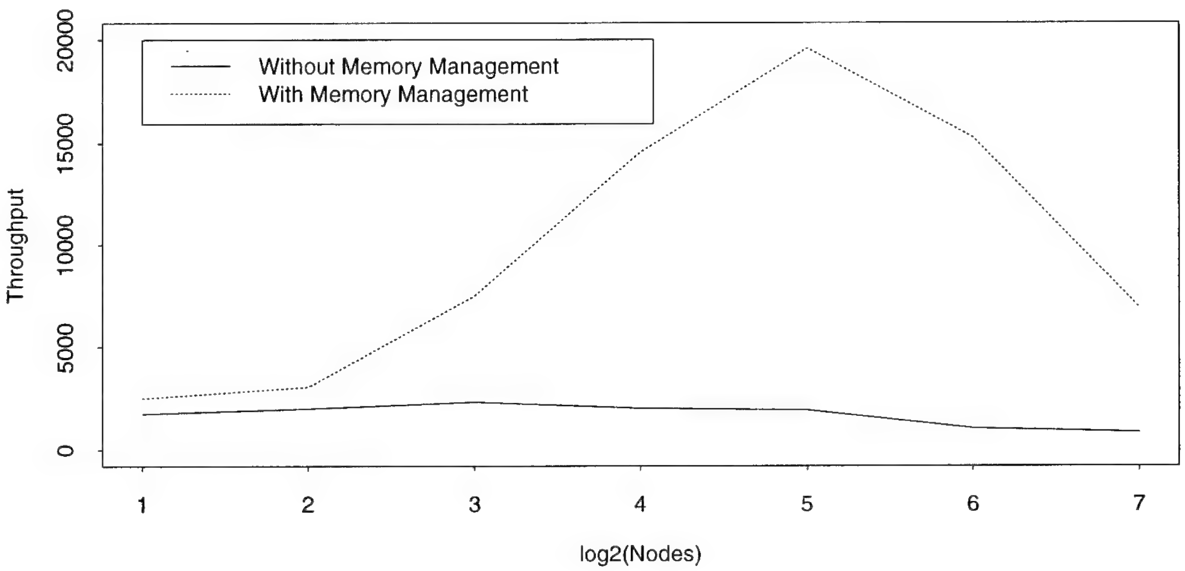


Figure 20 Impact of serialization of memory management system calls

Threads Synchronization

Since our system model is implemented as a large number of pthreads, interacting asynchronously by exchanging messages, synchronization among threads can get prohibitively expensive. Consider an implementation where a mechanism writes directly to the input message buffer of another mechanism (see previous section). If many threads attempt to access a buffer within a short time and the thread that is currently holding the buffer lock is scheduled out, then all threads waiting on the lock have to wait for this thread to run again. This happens more often as the number of threads (system size) increases.

The use of the *communication kernel* provides one way of reducing expensive thread synchronization. First, no more than two threads wait on a buffer. Further, when a network mechanism blocks on accessing one of its buffers, it does not have to wait for a long time, because the thread holding the lock is a communication thread, which is never scheduled out.

Figure 22 shows the performance difference between using and not using a communication kernel. In this experiment, 4 processors are dedicated for communication. For the system that does not use a communication kernel, the performance is good for a small system size, but drops very rapidly when the system size exceeds a certain threshold (32 nodes in this case). On the other hand, the system that uses a communication kernel exhibits stable performance up to 168 nodes. In addition, if the system size is larger than 64 nodes, the throughput of the system that runs on a communication kernel is always better than the throughput of the system that does not.

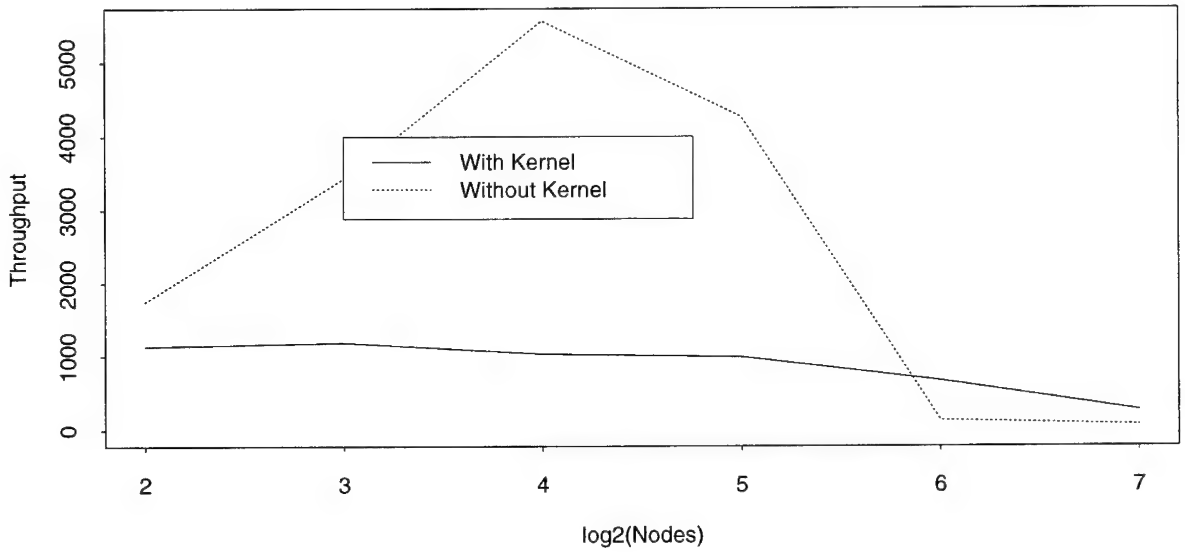


Figure 22 Impact of using communication kernel for emulation

In the next two experiments, we investigate how performance of the TCS is influenced by the amount of resources allocated to the communication kernel. In the first experiment, we fix the size of the system and vary the number of communication processors. In the second experiment, we fix the number of communication processors and vary the system size.

In the first experiment, the size of the system is 64 nodes and the number of processors allocated to communication is increased from 1 to 32. Figure 23 shows that the throughput of the system increases almost linearly with the system size. From this measurement, we see that the characteristics of the emulation system is such that more resources (up to 32 processors) can be allocated to communication (and less to computation) to achieve better overall system performance.

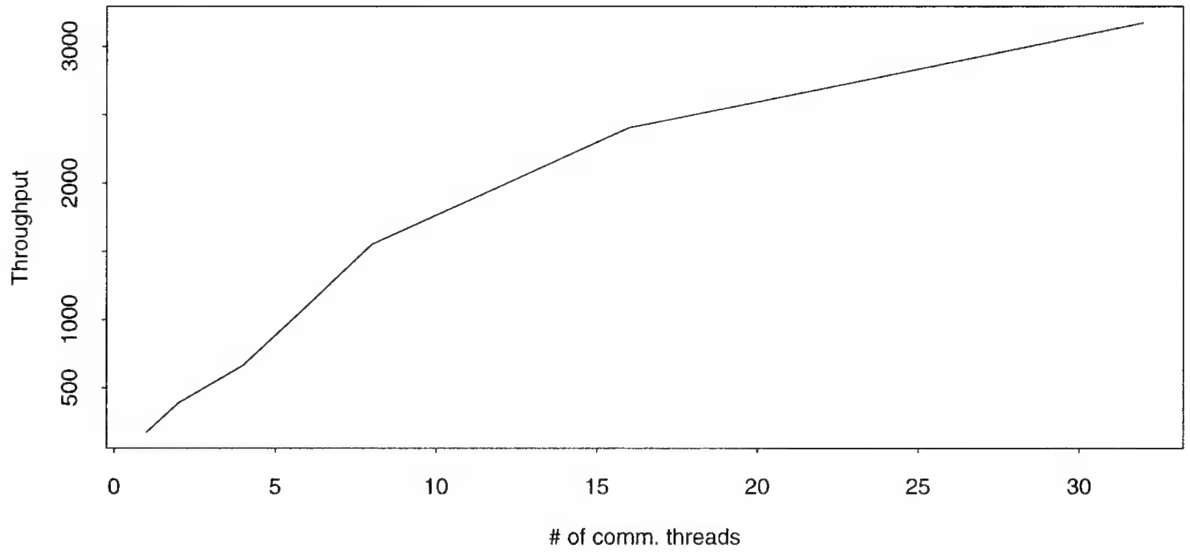


Figure 23 Impact of increasing communication resources (constant system size)

Figure 24 shows the result for the second experiment. We run the TCS with three different numbers of communication processors: 4, 8, and 16. The throughput of the system improves significantly when the number of communication processors increases from 4 to 8. However, when increasing the number of communication processors from 8 to 16, the performance gain is significant only up to 32 nodes. We attribute this drop in performance gain to two factors. First, as the number of communication threads increases, the frequency of them trying to access locks that have already been acquired increases. Hence, on the average, a communication thread has to perform more tries to acquire a lock before getting access to one. Second, more resources (8 CPU) are dedicated to communication and less to computation. Therefore, for large systems, the gain in communication performance is partially offset by the loss in computational performance.

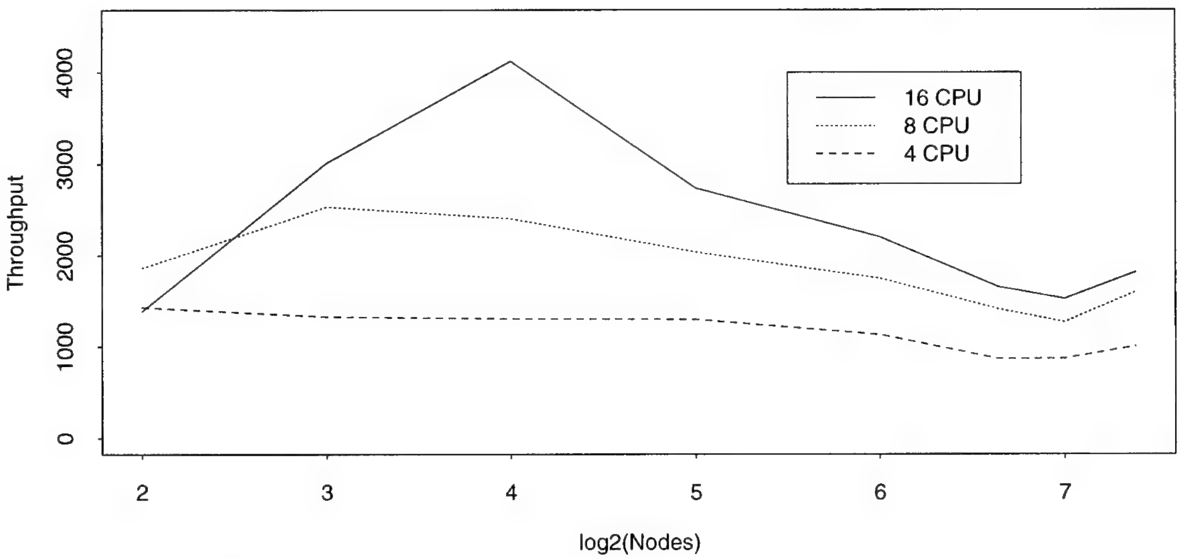


Figure 24 Impact of increasing system size (constant communication resources)

Context Switching Overhead

The overhead of context switching can be significant, if the amount of work done by a thread between context switches is small. In our application, this happens when a consumer, waiting on an empty buffer, wakes up and finds only one message (or very few messages) in the buffer. As the time it takes a thread to service a message is short in our application (of the same order of magnitude as a context switch), the thread finishes its work in a relatively short time, and waits on the empty buffer, causing another context switch.

We influence thread scheduling, by using conditional variables in a novel way, so that the amount of work done by a thread between context switches is increased. When a consumer waits on a conditional variable, instead of waking it up when there is one message in the buffer, it is woken up only if there are x (x>1) messages in the buffer (in implementation terms, we perform a "cond_broadcast()" call only if the number of messages in the buffer is equal to x). The same strategy applies to a producer. As a result, when a consumer or a producer is put in the run queue, there are at least x messages in the buffer.

Figure 25 shows the performance gain by influencing thread scheduling in such a way. The solid line graph shows throughput for x = 10, and the dotted line graph shows throughput for x = 1. The gain in performance is significant for system size of 64 nodes or more. In this experiment, the system uses application-level memory management and runs on a 4 processor communication kernel.

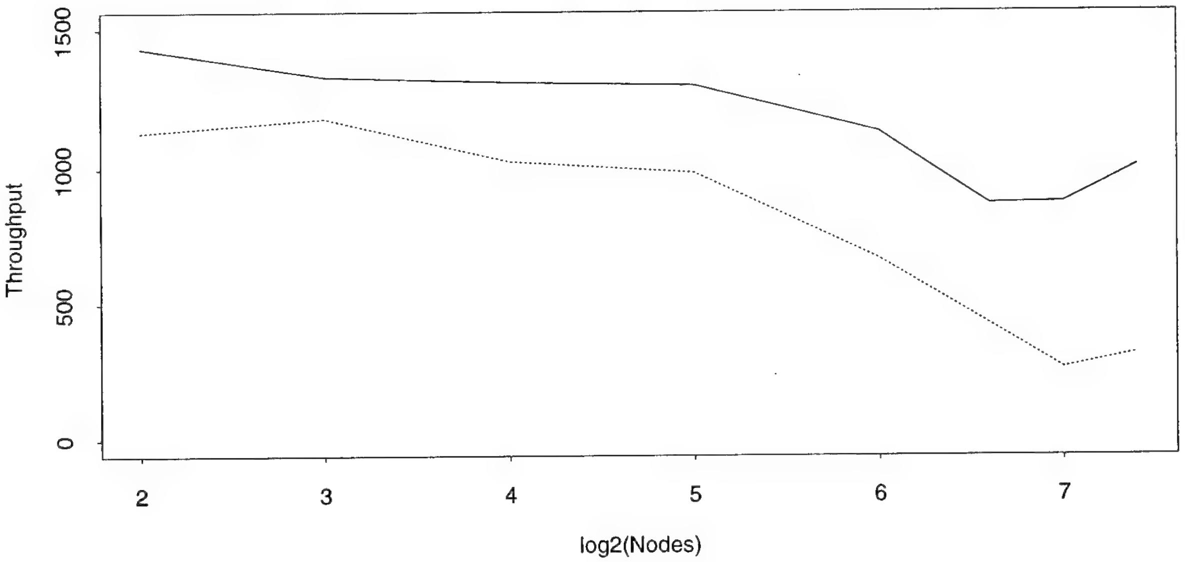


Figure 25 Impact of influencing thread scheduling

4.3 Porting the Emulation Platform from KSR-1 to SP-2

The IBM SP2 has a distributed memory architecture. Each node consists of a 266-MFlops POWER2 processor, which runs its own copy of the AIX operating system, and has between 64 and 2048 Mbytes of memory. The parallel programming environment we use is based on the Message Passing Interface (MPI) [MPI94] package. As opposed to the KSR-1, communication among processes on different processors is possible only through message passing. The benchmark for interprocess communication is therefore not dominated by memory access time and synchronization, but by the time delay to send messages from one processor to another. For the current MPI implementation on the SP2, sending a 200-byte message from one node to another takes about 50ms. Since communication on the SP-2 is off-loaded to a communication processor, communication and computation can often overlap.

The major differences between a shared memory machine and a distributed memory machine are summarized below.

	Shared Memory Machine (e.g. KSR1)	**Distributed Memory Machine (e.g.SP2)**
Address Space	Shared Address Space.	Address space is not shared.
Inter-Process Communication	Shared Memory Access (Mutual Exclusion).	Message Passing (e.g. MPI).

Table 5: Shared Memory vs. Distributed Memory

Since the programming model for the simulation kernel remains the same, much of the code can be reused. What needs to be changed, however, are the ways objects are referenced (and named) and how local clocks are synchronized.

Tasks	Shared Memory Machine (e.g. KSR1)	Distributed Memory Machine (e.g.SP2)
Object References and Location	Object references are pointers. Shared address space makes locating another object trivial.	Mapping from name to location (which processor) is needed.
Clock Synchronization	Local clocks are easily synchronized using a single shared memory variable.	A clock synchronization protocol is needed to estimate the difference between local clocks.

Table 6: Task Involved in Porting the Emulation Platform

Object references are different between the two platforms because on the KSR-1, the programmer assumes a global address space. All objects can thus be referenced through object pointers. However, on the SP-2, all objects have to be explicitly given globally unique names. These names are written into the messages exchanged so that the simulator kernels have sufficient information to route them to the correct destinations. Therefore, a naming scheme has to be devised and all objects given unique names when the simulation system initializes. Currently, the unique name is an integer constructed out of the object class type and the integer index of an object within its class type. It is assumed that there are never more than 1024 instances of an object per class.

Clocks among all processors need to be synchronized because we are running real-time simulation. On the KSR-1, this is performed by designating a “master” processor whose local processor clock serves as the common clock. A clock object is created and is “pinned” to run only on this processor and it writes continuously onto a known clock variable the value of its local clock. When an object needs the common clock time, it simply reads this clock variable. Since reading remote memory is on the order of 10ms, if the memory is available in the same ring 0, objects in the simulation system can have their clocks synchronized to within 10ms. On the SP-2, the approach is to first estimate the clock difference between a “master” processor and all other processors by exchanging many short messages and keeping track of the round trip delay time. Knowing this difference allows each processor to estimate the clock on the master processor by adding an offset to its local time. Assuming that the drift of the processor clock is small within the duration of the experiment (the experimental time for real-time session is usually not more than an hour), the clocks can synchronize to within 100ms or better.

5 Prototyping the VPN Architecture

5.1 The Prototyping Approach

Two main tasks are involved in the development of a network control system: the development of a software system and the design and analysis of control algorithms. The first task focuses on the software engineering aspects that satisfy the system requirements. The second concentrates on developing control functions that meet performance objectives. So far, these tasks have been carried out separately, using different tools and environments. As a result, performance evaluation studies concentrate on control algorithms, missing the evaluation of the dynamic behavior and complex interactions that take place in the network control system. A thorough evaluation of the performance characteristics of a network control system has to take into account both system design and control algorithms. This implies that the processes of system development and performance evaluation have to be combined and performed in an integrated way.

Our approach to developing a network architecture is to build a prototype of the system design, i.e., an "easily built, readily modifiable, ultimately extensible, partially specified, working model of the primary aspects of the proposed system" [CON95]. The prototype is developed incrementally, and it is exercised, validated, and revised after each development step. The prototyping paradigm provides benefits in our application domain: a prototype allows us to evaluate the characteristics of the system before the implementation phase on a testbed. Furthermore, by exercising the system, we become aware of additional, previously unknown or unnoticed system requirements. Also, we discover unanticipated side effects of design decisions (which actually happens quite often).

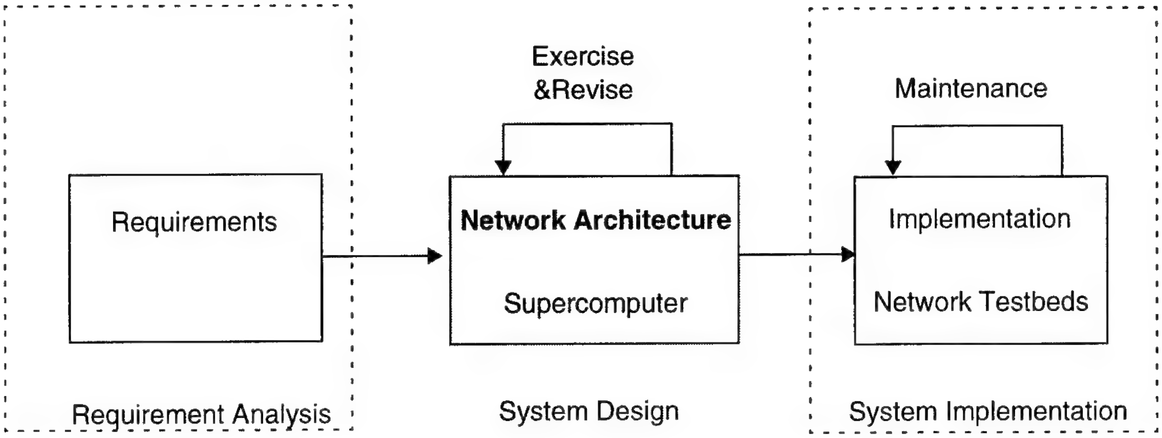


Figure 26 Software development process

Figure 26 describes the typical phases of a software development process and illustrates the methodology followed in this report. Developing software starts with analysis activities, followed by design activities, followed by the implementation and coding phase. The analysis phase determines the requirements of the system, while the system design phase produces a model that describes how these requirements can be met. This model is independent of the target hardware and software environments. During the implementation phase, code is generated to realize the system design with the required functionality and performance characteristics on the target platform.

We have built a network emulation platform[1] which allows us to develop and evaluate prototypes of network control systems. This platform includes a massively parallel machine which emulates an executable model of the architecture and a graphics workstation which visualizes the state of the network in real-time and allows us to change control and configuration parameters of the emulated system during run time. The emulation platform runs on two types of parallel computers--the KSR-1 and the IBM SP2. The front end is implemented on an SGI graphics workstation. This platform is currently used in several projects in our laboratory which are aimed at developing and evaluating network architectures [CHA96a, CHA95a, PAC95].

The use of an emulation platform allows us to overcome the many limitations of network testbeds. Testbeds have been used to demonstrate the feasibility of providing high-speed connec-

[1] The emulation platform is not part of the contract deliverable but its design is summarized here because of its influence on the VPN architecture design.

tivity and multimedia communication capabilities among small groups of users [CLA92, FRA92, STO92]. Unfortunately, the results produced on such a testbed are generally restricted to a specific system configuration and cannot be used to predict the behavior of the system when the number of users, services, and network nodes is increased. Furthermore, it is hard to perform experiments on network testbeds because of the intrinsic difficulty in instrumenting and monitoring a distributed system. Finally, it is hard to quickly modify a system on a testbed, since there is generally no homogenous development environment available.

Compared to a network testbed, tasks on a parallel machine can be easily monitored and controlled, components can be easily added or modified, and the system size can be increased, limited primarily by the processing resources of the parallel machine.

Our platform meets important requirements for building prototypes. First, it gives us a homogenous programming environment with base object classes to build network controllers and with support for communication among controllers. Second, it provides us with flexibility for testing purposes. Traffic generators, for example, allow us to study the system under different load patterns. Aspects of system configuration, including the network topology, are defined in files that can be changed from run to run. Third, our platform supports interactive control of the prototype during run time and dynamic visualization of the system in real-time. We can run what-if scenarios and we can observe dynamic phenomena such as oscillation of the network state or transient behavior. Fourth, network control systems need a large amount of computational resources to run in real-time when the system size increases. The multiprocessor we are currently using gives us the capability to allocate a sufficient number of processors (technically, up to 512) to perform scalability experiments.

Finally, in order to carry out performance evaluations, we need to control the execution time of the prototype system. The classical way to achieve this is to run the prototype as a (parallel) discrete event simulation, which allows us to model delays. Following this approach, we built a parallel simulation kernel as part of the emulation platform. To facilitate the construction of prototypes, we designed the platform in such a way that the simulation is transparent to the developer of a network architecture. This means that a user of our platform needs little simulation knowledge (see Section 5.4).

NEST [DUP90] is another platform developed for prototyping networks. It provides an integrated environment that supports system development and simulation. This environment allows developers to use the same set of tools, such as compilers or editors, for system development and simulation, and to re-use some code segments. However, this platform is built around a single processor simulation server which does not provide the necessary performance we require for our evaluation studies. (For a survey on network simulation tools, see [COM94].)

5.2 Design of the Emulation Platform

The emulation platform consists of four building blocks: *parallel simulation kernel, emulation support, real-time visualization and interactive control,* and *emulated system* (Figure 27) . The emulated system and emulation support modules consist of a set of objects that communicate by exchanging messages, using functions provided by the simulation kernel. The emulated system module represents the prototype of the network control system under evaluation.

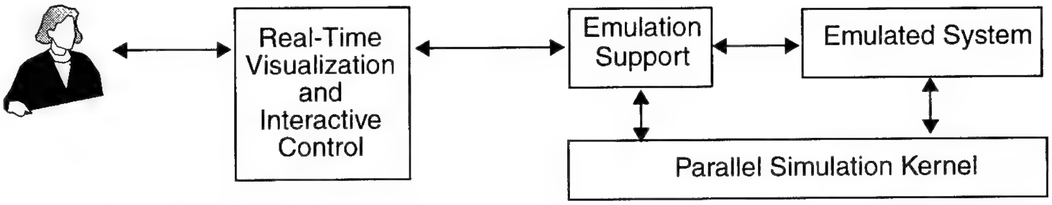

Real-Time Visualization and Interactive Control
Emulation Support
Emulated System
Parallel Simulation Kernel

Figure 27 Building blocks of the interactive emulation platform.

The *simulation kernel* controls the execution of these objects and ensures that messages are processed in the correct order. It implements the paradigm of parallel discrete event simulation (PDES) [FUJ90], using a variation of the window-based synchronization protocol described in [NIC93]. In order to support real-time visualization and interactive control of the emulated system, the kernel controls the progression of the simulation time, constraining it by the progression of the processor time. The design of this kernel is discussed in [CHA96b].

Objects that interact with the parallel simulation kernel require minimal knowledge about the kernel--mainly how to send and receive messages. Therefore, the design of the emulated system follows the same rules as the design of network controllers that run on a real network platform. The major difference is in how the interaction among controllers is realized. In the emulated system, interaction is performed by the simulation kernel. In a broadband network environment, for example, the exchange of messages is provided by a signalling system.

The module for real-time visualization and interactive control contains a graphical interface which provides 3-D visual abstractions of the system state. The state of the emulated system is continuously refreshed, which allows a user to follow the system dynamics on the interface. The user is able to control the behavior of the emulated system by changing parameters through the interface.

The emulation support module coordinates the exchange of control and monitoring messages between the graphical interface and the emulated system. It reads the states of the emulated system, and performs filtering and abstraction operations before making the information available for visualization. Control information from the user is mapped to a set of control parameters that are interpreted by the emulated system.

The design of the emulation platform along the four building blocks described above serves the purpose of portability and re-usability. The platform runs on two types of supercomputers--one with a shared memory architecture and the other having a distributed memory architecture. The two implementations differ in emulation support and types of simulation kernel modules. The emulated system module needs only minor changes when porting the software from one computer to the other; the module for real-time visualization and interactive control needs no modification.

5.3 The Parallel Simulation Kernel

The simulation kernel controls the execution of simulation objects and routes events from one object to another. Objects that require services from the kernel are defined as a subclass of `SimObject`. Each `SimObject` (or any subclass of this object) has a `processEvent` method with an input parameter of type `Event`. Attributes of an `Event` include a time-stamp indicating when the event can be processed, object identifiers of source and destination, and an event-type tag. An object generates an event by invoking the `sendEvent` method which has an output parameter of type `Event`. The simulation kernel receives this event, routes it to the destination, and invokes the `recvEvent` method of the destination object.

Figure 28 shows the realization of the parallel simulation kernel. It is implemented as a set of local simulation kernels, each running on a separate processor. A local simulation kernel controls the execution of objects allocated to its processor. Routing of events is performed using a copy of a global object allocation table which is set up during the initialization phase of the simulation.

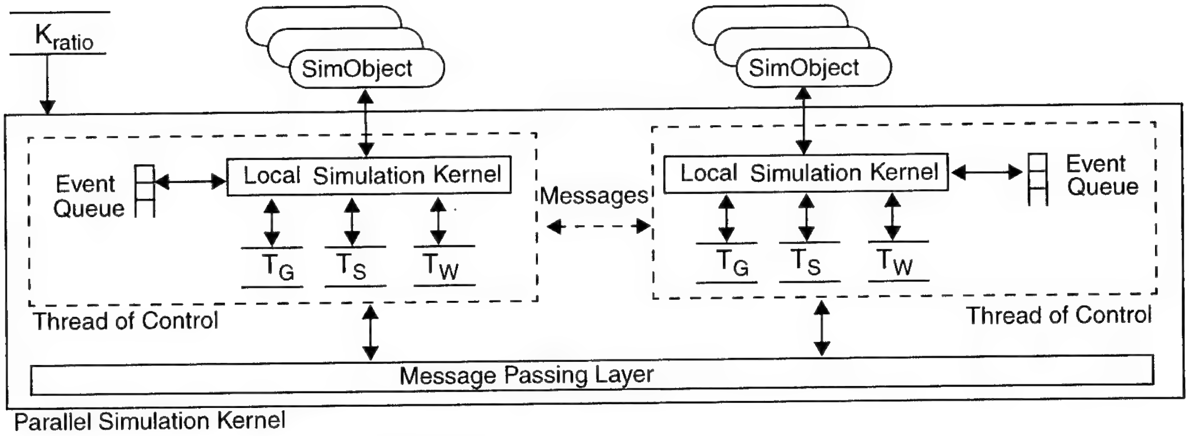


Figure 28 Realization of the Parallel Simulation Kernel.

The simulation kernel controls object execution using a global time T_G. It delivers an event to an object for execution only when the time-stamp of the event is smaller or equal to T_G. Otherwise, the event is buffered in the event queue.

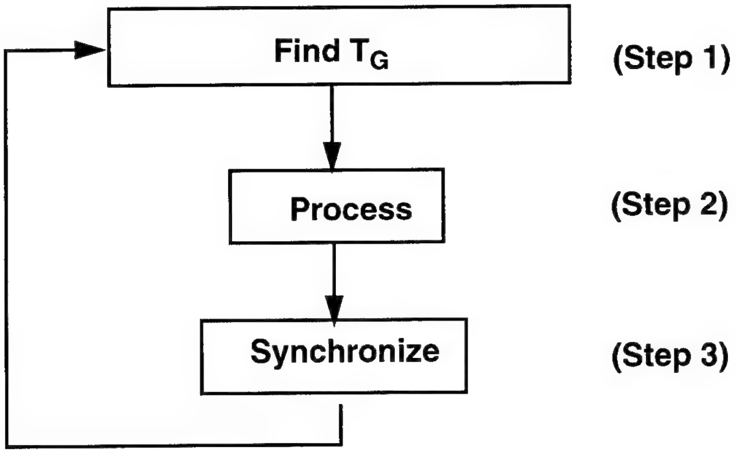
T_G is computed as the minimum of T_S and T_W, which are explained below. T_S is the simulation time as computed by the causality-control protocol, which is based on the window-based synchronization protocol described in [NIC93].

T_W, the scaled wall-clock time, is a linear function of the processor time taken from a reference processor, i.e., $T_W = \text{Ptime} \cdot K_{ratio}$, where P_{time} denotes the processor time and K_{ratio} is a control parameter that can be changed by the user.

Performing the simulation in the above described way allows us to synchronize the evolution of the simulation time with a linear function of the processor time. Our system is therefore capable of reproducing and visualizing the dynamics of the real system, with the time scaled by a factor. By changing K_{ratio}, we can control the speed at which the simulated system state evolves. For example, we can slow down or “speed up” the simulation. (Speeding up the simulation is obviously constrained by the performance characteristics of the hardware.)

By processing an event only when its time-stamp is smaller than T_S -- i.e., when no causality violation can occur -- we follow the conservative approach to parallel simulation. We eschew the optimistic approach, which requires roll-back and state saving mechanisms, mainly because that approach makes it difficult to build simulation objects independent of the simulation kernel [NIC95]. Our objective is to have a clean separation between the emulated system and the simulation kernel, rather than improving the performance of the simulation kernel. (It is argued that, in general, optimistic simulation allows for better performance of a parallel simulation kernel [FUJ90], although our application falls into a class where conservative simulation exhibits a comparable speedup [NIC95].)

On a function level, the task performed by a simulation kernel can be described by a loop consisting of three steps. In step 1, the set of local simulation kernels runs a synchronization protocol to compute T_G. In step 2, each logical process is allowed to process all events with time-stamp not later than T_G. In step 3, when all events have been processed, the set of local simulation kernels has to make sure that all events sent previously have been received before they proceed back to step 1. Synchronization in step 1 and 3 are performed using MPI primitives on the SP-2. In step 1, a call to MPI_Allreduce() is used to find the minimum time-stamp of messages received by all logical processes. In step 3, there is one call to MPI_Barrier() and then one call MPI_Allreduce() to check that all messages sent are received. The overhead introduced by these synchronization operations are measured for 4 processes, each running on a different processor, calling MP_Allreduce() or MPI_Barrier() in a tight loop. The result shows that MPI_Allreduce() takes on the average 0.2ms to execute, while MPI_Barrier takes 0.1ms to execute.



Benchmark of some primitives (4 CPUs)

In (STEP 1)	0.2 ms
In (STEP 3)	0.3 ms

Figure 29 Performance of Simulation Kernel

The performance of a 50 node experiment running on the simulation kernel is shown in Figure 30. In this measurement, we ran the traffic control system, which includes traffic generators, connection managers, routers, and admission controllers. There are more than 250 objects in the simulated system. The number of processors used in the simulation is increased from 2 to 48, and the number of events processed per second by the simulator is recorded. The figure shows a reasonable increases in performance when the number of processors increases, though as expected, the increase in far from linear.

An interesting question one can ask is how many processors are needed such that the simulation can run in real-time. In other words, how many processors are needed such that the simulator can process events fast enough so that $K_{ratio} = 1$ (Ptime = T_w), and $T_G = T_W = T_S$. As an approximation, we assume that events are proccessed at a uniform rate. Thus, the event processing rate required for real-time simulation is taken to be the ratio of the total number of events processed and the total simulation time. In Figure 30, the horizontal line drawn across the figure indicates the average event processing rate required for real-time simulation. The interaction between this line and the speedup curve gives the number of processors needed. For this particular system configuration, about 26 processors are needed.

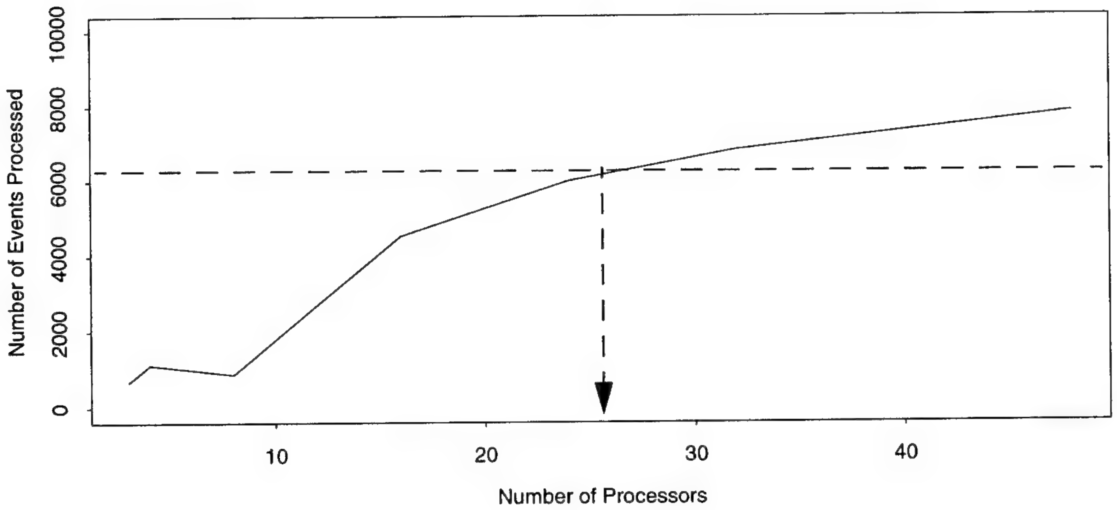


Figure 30 Scalability of a 50 node experiment running on the simulation kernel

5.4 Emulation Objects

Emulation objects, i.e., objects in the emulated system, are instances of subclasses of *SimObject*. The purpose of the base class *SimObject* is to hide the simulation functionality from the user by enforcing a clean separation between *SimObject* and the emulation objects.

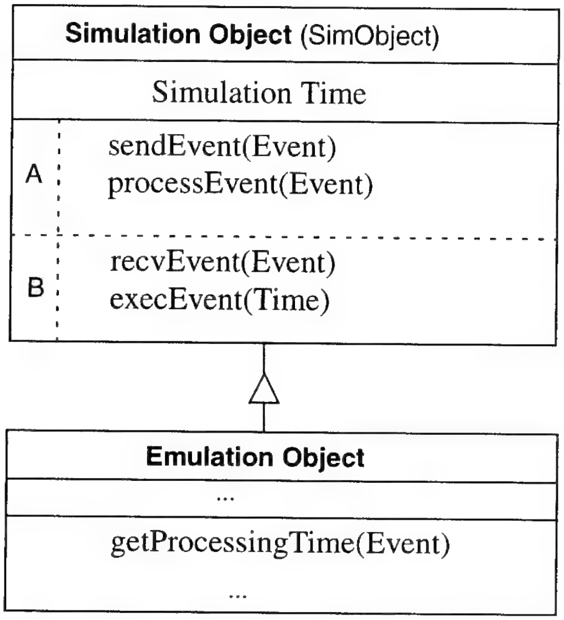


Figure 31 Inheritance relationship between simulation and emulation object classes.

Figure 31 and Figure 32 follow the notation of [RUM91]. Each object is described by its name (e.g., Simulation Object), followed by attributes (e.g., Simulation Time), and finally the methods associated with the object (e.g. sendEvent and processEvent). As shown in Figure 31, SimObject has two groups of methods. Group (A) consists of methods that are relevant to the emulation objects. The interface *sendEvent*() is a generic function used by all emulation objects to initiate the sending of messages. The interface *processEvent()* is used for interpreting messages that have been received, so that the appropriate methods in the emulation object are invoked. (This function is similar to the stub code function for RPC systems.) This interface is different for each class of emulation objects. The interface *getProcessingTime*() returns the time needed to process a particular event, drawn from a probability distribution that models the processing time. This interface is also different for each class of emulation objects. Group (B) consists of methods necessary to support a conservative approach to parallel simulation. These methods are common to all emulation objects. They are visible only to the simulation kernel and are not accessed by emulation objects.

In order to add a new emulation class to the emulated system, the user needs to specify the implementation of two methods, *processEvent()* and *getProcessingTime()*. The rest of the methods are inherited directly from *SimObject*. Therefore, the overhead of using our platform over a general purpose computing platform is that two additional methods need to be provided. Since in order to run an object in a message passing environment, a method equivalent to *processEvent*() is required anyway, the additional interface needed when writing an emulation object which runs on our platform is the method *getProcessingTime*().

Our design is based on the conservation approach to parallel simulation. This enables a clean separation between simulation and emulation model, because the designer of an emulation object needs minimal knowledge about the underlying simulation model. If an optimistic approach is adopted, the methods (B), which implement a conservative simulation protocol, must be replaced by a set of methods that implement an optimistic protocol. (The methods (A) remain.) In this case, however, the designer of an emulation object has to deal with state-saving that is needed if roll-back occurs. The designer has to specify, for each class of objects, the states to be saved. This is a non-trivial task, which requires knowledge of the simulation model in order to obtain good simulation performance. In this sense, the conservative approach provides an easier-to-use simulation environment, compared to the optimistic approach [NIC95].

A fundamental abstraction in the domain of network control and management is the concept of a *resource*. We model a resource as an object characterized by the attributes `capacity` and `operating point`, which indicates the utilization of the capacity. The capacity of a resource defines an upper bound for the operating point. Resources are modeled as finite capacity queues without buffers. A resource controller regulates access to a resource. It receives requests for access to the resource and decides whether to accept or reject the request, based on the capacity, the current operating point, and the control policy. We model resource controllers as emulation objects with methods that allow for resource negotiation and generic monitoring (see Figure 32).

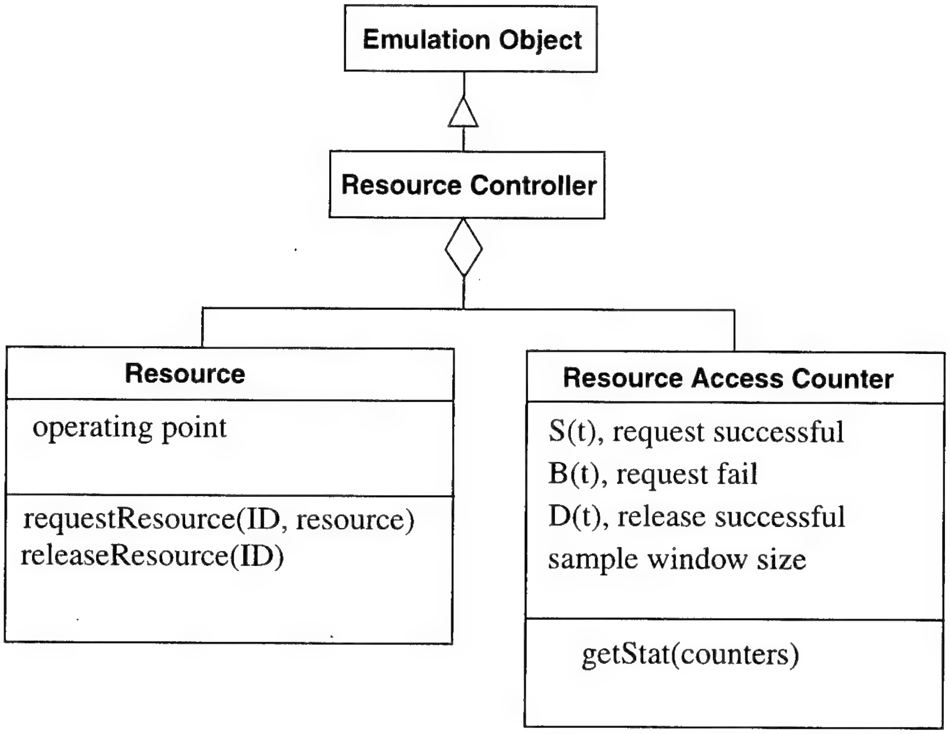


Figure 32 Generic monitoring and access of a resource controller.

A monitoring operation on a resource controller object returns the following event counters:

- S(t) the number of successful requests at time t.

- B(t) the number of unsuccessful requests at time t.

- D(t) the number of departures at time t.

Statistics relating to a resource controller are derived from these counters. For example, N(t), the number of active requests at time t is given by S(t) - D(t), and the average blocking between the time interval t_1 and t_2 is $(B(t_2) - B(t_1))/ (t_2 - t_1)$. The window size of the samples kept in memory determines how sensitive the statistics are to short-term variations.

Emulated Objects communicate via asynchronous message passing. For example, in one of our network prototypes, messages are exchanged among connection managers and link admission controllers during a connection setup. Logically, a mailbox is associated with each emulation object. If object A wants to send a message to object B, it invokes a send function that inserts the message into B's mail-box. Conversely, object B receives messages from other objects by invoking a receive function that removes the messages from its mailbox.

Our platform supports asynchronous message passing, because it is a flexible communication paradigm. It can be used to emulate other paradigms, including synchronous message passing, generative communication, and remote procedure call (RPC) [AND91]. Furthermore, the Signalling System No. 7 [ADB90], which is a part of the telecommunications networks' control systems, also uses message passing as the communication paradigm.

5.5 Emulation Support

Figure 33 shows a data-flow diagram of the software architecture for the support of real-time visualization and interactive control. Its design follows the monitoring-control paradigm. Monitoring is realized as a continuous activity, whereby a stream of data produced by the network emulator is consumed by the operator interface in the graphics workstation. Interactions among components involved in the monitoring process are asynchronous, via reading and writing shared objects, which allows them to run on different time scales. Control operations, on the other hand, are event-driven and are based on the client-server paradigm.

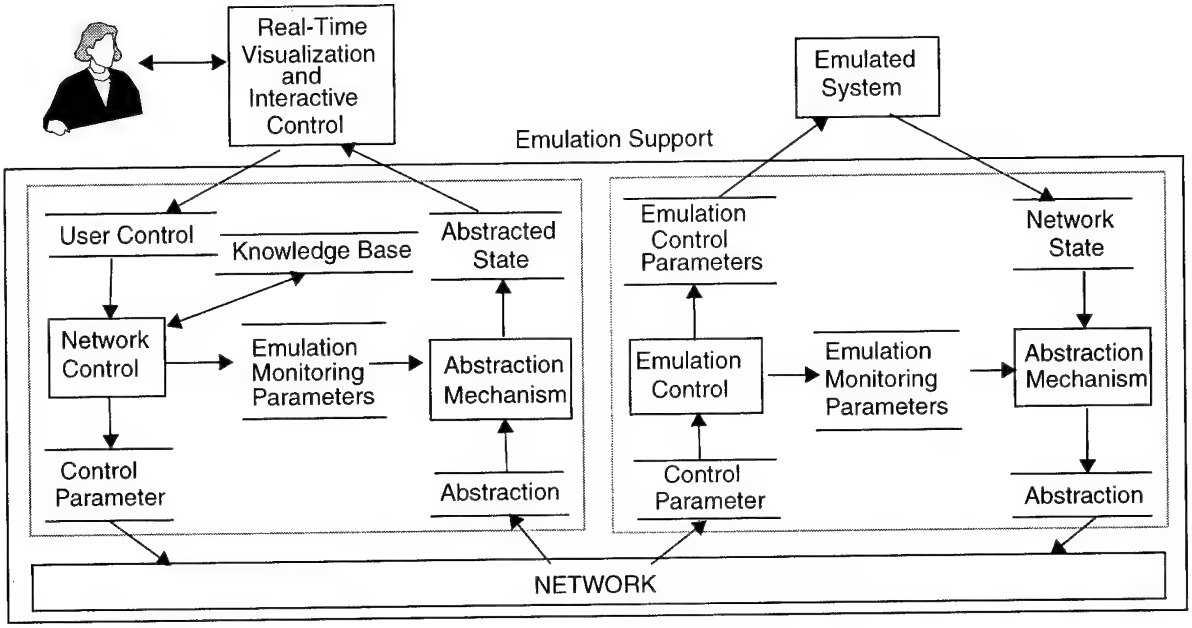


Figure 33 Software architecture for the Emulation Support module.

There are two types of control operations: (1) *emulation control operations*, which alter the behavior of the emulated system and are executed by the network emulator, and (2) *monitoring control operations*, which affect the monitoring activity. *Emulation control operations* include changing the input load on the system and modifying the control policies with which the control mechanisms run. Monitoring control operations tune components in both the emulation system and the manager station. They regulate the part of the network state that can be visualized on the manager station, as well as the amount and the granularity of data that is carried over the network, by, for example, adjusting the sampling interval and the size of the monitoring window. Inside the manager station, the information collected from the various network objects can be abstracted, correlated and displayed on the screen in different ways. By varying the quantity and granularity of information collected and sent over the network, our system can be tailored to different computing and network platforms.

5.6 Real-Time Visualization and Interaction

The design of the Real-Time Visualization and Interactive Control module focuses on the selection capabilities, high-level control primitives available to the operator, and visual representation of network states. These modules allow an operator to try what-if scenarios, invoke specific network services during run-time, emulate network management operations, and visualize the effect of these operations.

Operations performed by the human operator through the interface are categorized into three groups. The first set relates to defining the input traffic characteristics for each network node. The second set enables a variety of monitoring functions. By selecting objects (switches, links, network regions, etc.) on the network map, together with the desired monitoring option, the operator can visualize, in real-time, various abstractions of the network state, including traffic intensities and network utilization. The third set of interface operations refers to changing management parameters, which allow operators to tune the behavior of mechanisms in the traffic control system.

This remaining part of this section gives an overview of the structure and the composition of the network management system consisting of a graphical user interface which allows interactive control of real-time services.

5.6.1 Processes and Tasks

The network emulation and visualization is done by different processes that run on different machines. The emulation and simulation of the network is computed on a parallel computer, while the visualization of the network state is done on an SGI workstation. The main process on the workstation is *Net3D*. Net3D starts the sender and receiver process, builds the user interface including the three-dimensional graphics and handles the user input.

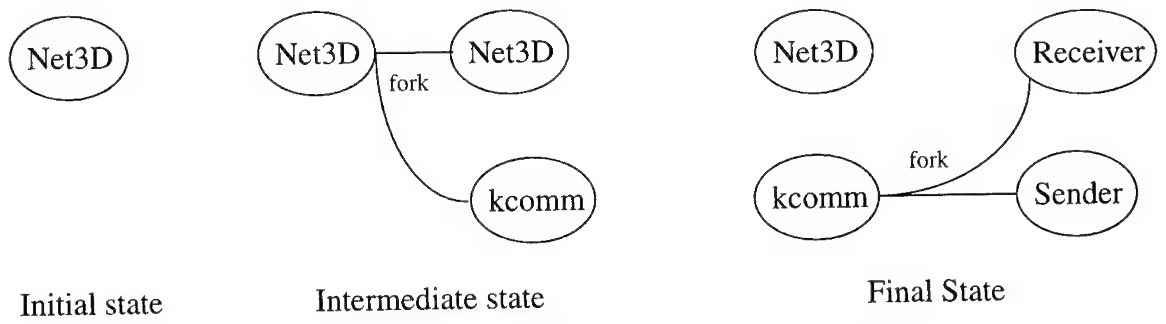


Figure 34 Generation of the processes in two steps

Figure 34 shows the phases of the initialization process. Net3D forks a child process *kcomm* which deals with the communication tasks. kcomm will be divided further into two processes, one for sending and the other for receiving data to and from the network emulator.

Communication between the kcomm processes (sender and receiver) and the emulator is performed using TCP/IP sockets. Figure 35 gives an overview. The sender/receiver processes and Net3D communicate over shared memory segments. There are three different segments corresponding to the different tasks the processes have. The statistics data describing the current traffic load, number of calls etc. are written by the receiver process into the shared memory segment for monitoring. Net3D reads from the shared memory and updates the graphics. Receiver and Net3D operate asynchronously using independent control cycles. The result is that the receiver can update the data more (or less) frequently than Net3D reads the data. When the control parameters are changed by the user, the new values and corresponding function tags are sent to the emulator by the sender process. The sender reads the values and the tag from the shared memory segment for control. Depending on the operation, the sender may write the values to a third shared memory segment, the parameter segment. If later the values of the parameters are needed to show the current state, Net3D can read the values from the segment and does not have to get the values from the emulator. So the parameter segment serves as a storage for the parameter values. The control parameter, for example, allows the user to change the values for the call arrival and departure rate and shows the current values.

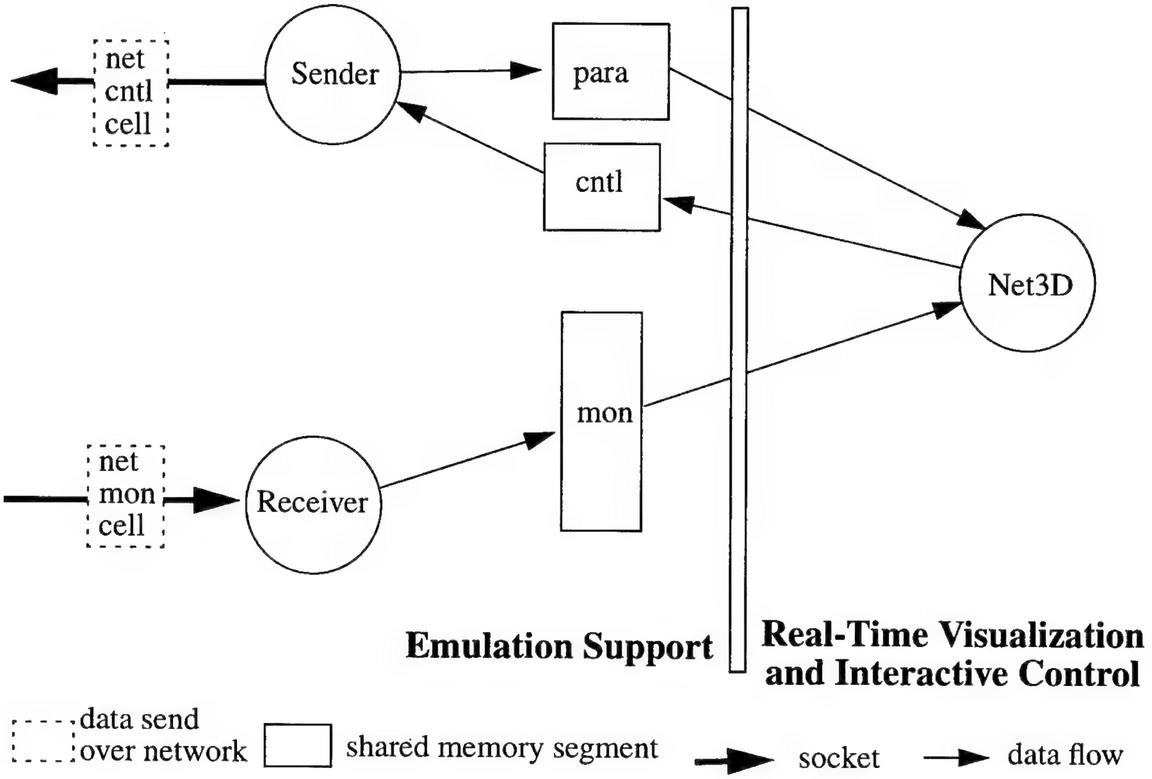


Figure 35 Processes and communication means

5.6.2 Modules and Classes

The system can be divided into two subsystems which correspond to the communication functionality on the one hand and the visualization functionality on the other. The visualization functionality consists of a part that builds up the menu and handles the user events and a second part that visualizes the network and the related statistics. The visualization subsystem is implemented in C++ resulting in a modular, abstract class system. The modules dealing with sockets and shared memory are written in C and rely on global shared variables, which leads to a system that is not as modular and abstract as the graphics subsystem. However, the advantage resulting from this approach is a high performance which is fundamental for real-time visualization. The module *smmgr.c* containing the functionality for the shared memory management and is used by both subsystems.

5.6.3 Sender and Receiver

The sender and the receiver are both implemented in *comm.c*. The functions *sendfunc()* and *recvfunc()* correspond to the sender and receiver respectively. After initializing the shared memory segments and the connections, the sender and receiver both go into a loop. The sender waits for requests to send to the emulator. The receiver waits for incoming data from the emulator. The exchange of data between the sender and Net3D is controlled by a handshaking mechanism using a control variable placed in the shared memory segment. After a write operation the Net3D process writes 1 to the control variable. The sender periodically tests whether the control variable has changed to 1. If it has, the sender reads the data and sets the control variable to 0. Net3D can only write again, after the control variable has been set to 0. This results in the two following function pairs:

```
int write_lock_shared_memory_segment(shared_cntl_t* addr)
{
    while (addr->write == 1) { sginap(10); }
    return 1;
}

int write_unlock_shared_memory_segment(shared_cntl_t* addr)
{
    addr->write = 1;
    return 1;
}

int read_lock_shared_memory_segment(shared_cntl_t* addr)
{
    while (addr->write == 0) { sginap(10); }
    return 1;
}

int read_unlock_shared_memory_segment(shared_cntl_t* addr)
{
    addr->write = 0;
    return 1;
}
```

sginap() is a IRIX specific system call and is the equivalent of the *sleep()* function available on most UNIX platforms. The argument to sginap() is specified in units of 10ms. Therefore, the call signap(10) will cause the process to sleep for 100ms. The data structure shared_cntl_t is described in more detail in section 5.6.4

5.6.4 Net3D

The shared memory management for *Net3D* is done in *smmgr.c* and in the class *StatisticsManager.* As mentioned above, three different shared memory segments are involved to change the management parameters of the simulated network. Net3D writes the corresponding tag and the required parameter values into the control shared memory segment. The results are then written to the parameter shared memory segment. The third shared memory segment contains the current statistics data updated by the receiver process and is read by the statistics objects. The corresponding data structure is shown in Figure 36. Bold names are type names, italic names are identifications of shared memory segments, and arrows represent pointers. The segment for monitoring (shmonid) is used as an array of blocks. Each of these blocks (of type shared_measure_t) belongs to a link and contains the values of the related parameters, as capacity, current load, etc. The global variable shared_mon_addr is used by the statistics objects to fetch the data. Each statistics object has an index that determines the pointer to the appropriate block in the segment. The assignment of the blocks (indices) to the links is done in the statistics manager, an object of type StatisticsManager.

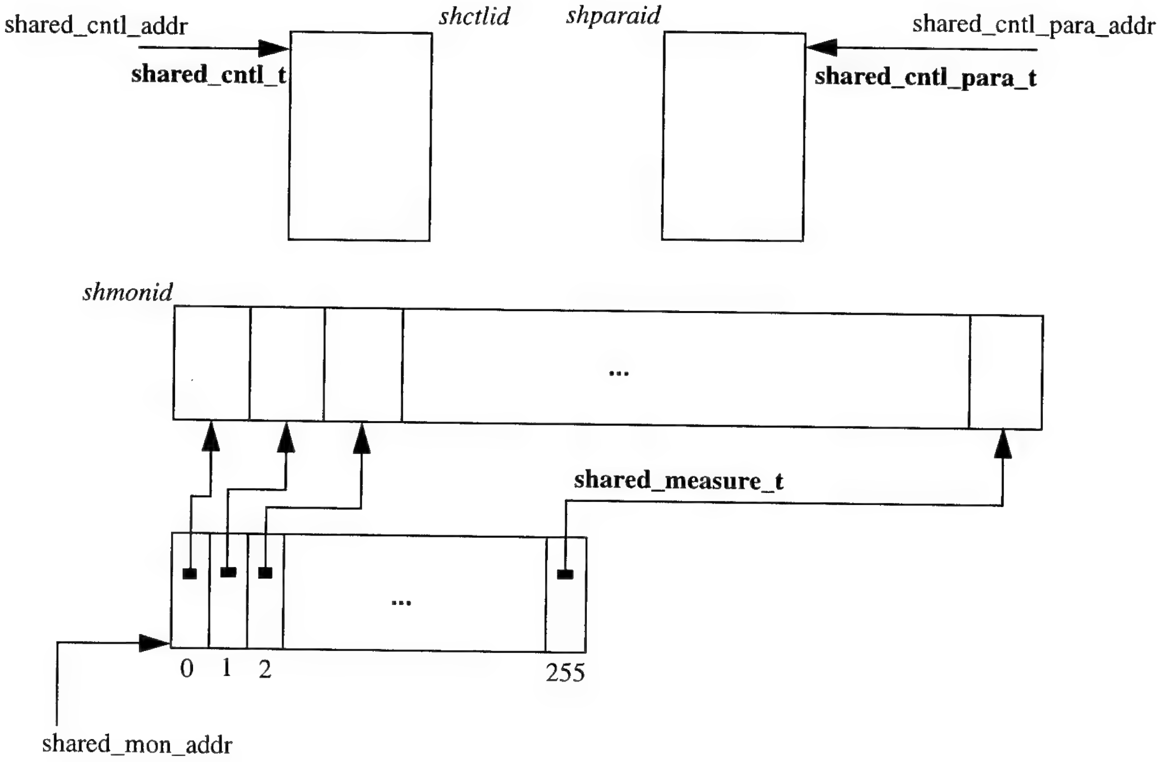


Figure 36 Shared memory data structure in Net3D

5.6.5 Network

The visualization part encompasses two different tasks. The first task is the visualization of the network consisting of links and nodes which can be composed to net regions. There are two layers visualized, one showing the virtual paths (VP) and nodes, the other showing the virtual path groups (VPG) and nodes. The data structure used to model the network is shown in Figure 37 and Figure 38. An object of the class from which the arrow starts has an object that belongs to the class at which the arrow points. A circle at the point of the arrow means that several objects are attached. The class *NetAppWindow* is the container class in the sense that an object of this class builds the whole datastructure needed to model the network and to provide the user interface.

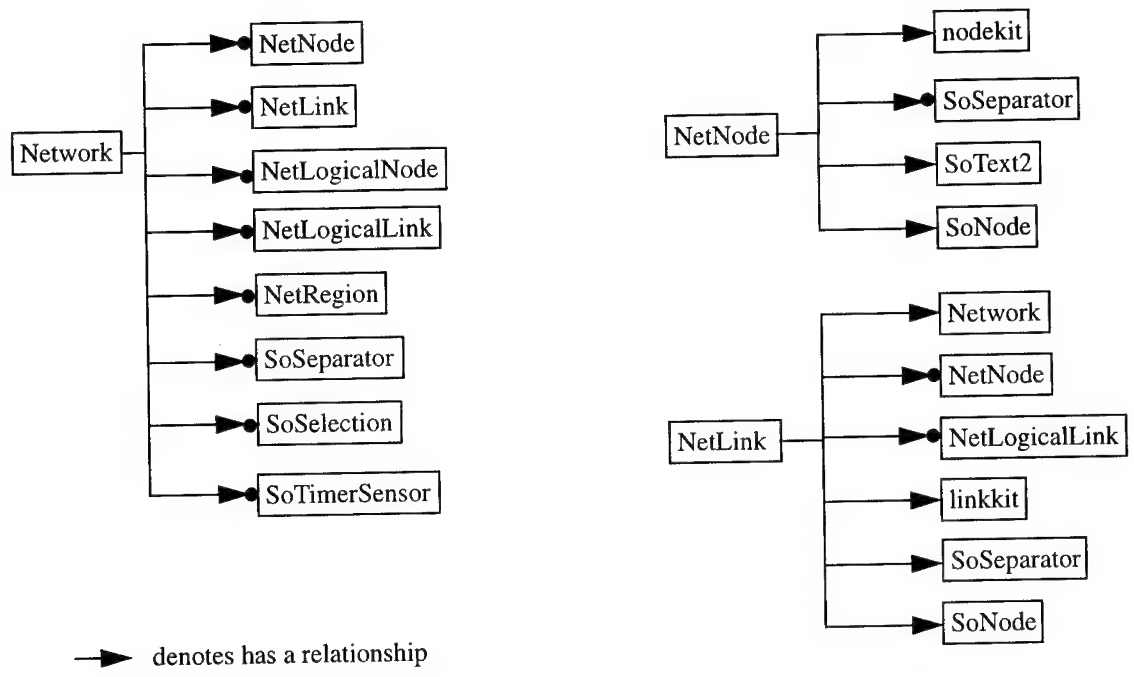


Figure 37 Data structure modeling the network (I)

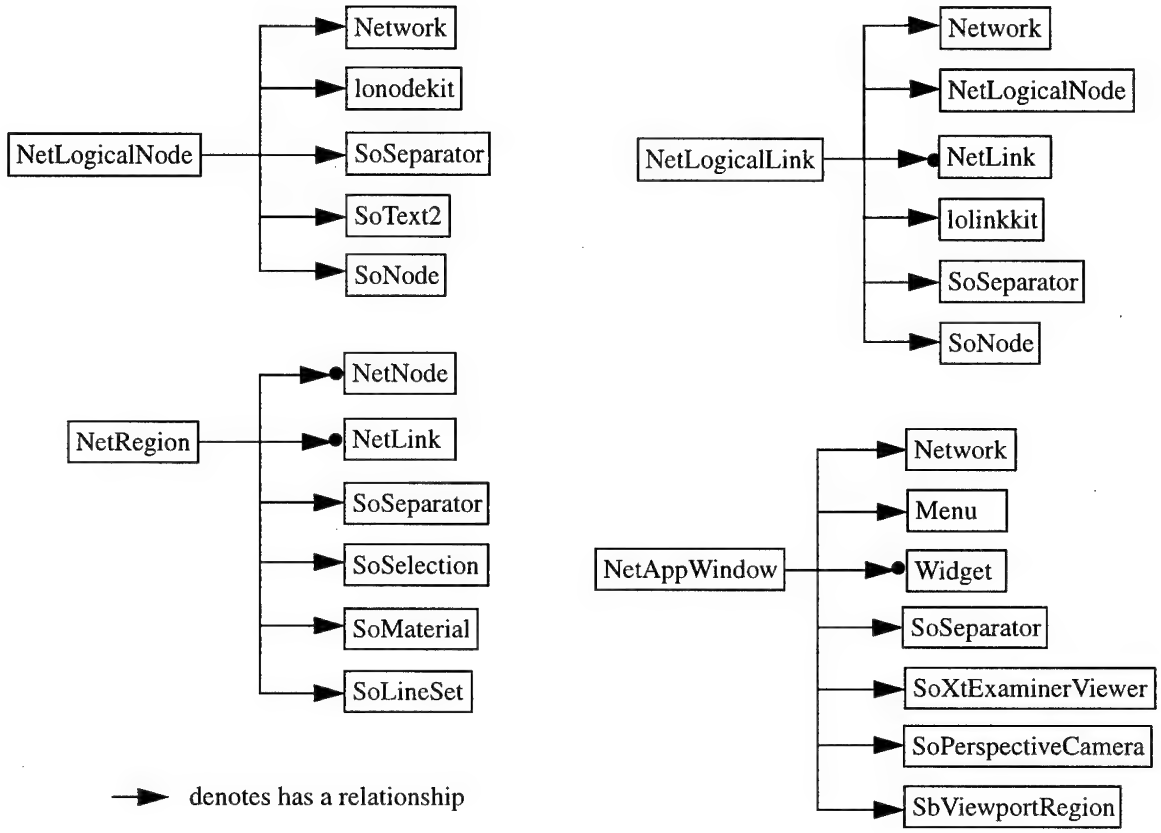


Figure 38 Data structure modeling the network (II)

The graphical objects are written using the OpenInventor shape kits. These are higher level abstractions for assemblies of elementary objects as lines, cubes, transformations, colors etc. Four different shape kits are used for visualizing the network, namely, *nodekit* (VPG -node), *lonodekit* (VP-node), *linkkit* (VPG-link) and *lolinkkit* (VP-link). The inheritance relation with the base class *SoBaseKit* is shown in Figure 39.

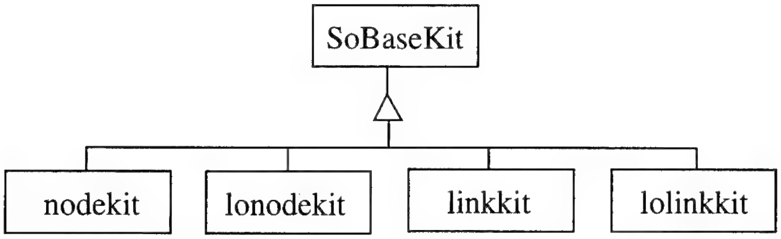


Figure 39 Inheritance relation of shape kits

The related files are *NodeKit*.[hC], *LogicalNodeKit*.[hC], *LinkKit*.[hC] and *LogicalLinkKit*.[hC].

5.6.6 Statistics

The second task is the graphical representation of the network state. There are various classes dealing with statistics. The inheritance graph is shown in Figure 40. The class *Stat* is an abstract base class (and therefore in italic). The numerical values are visualized by three-dimensional cylindrical bars or as curves and points in a 3D coordinate system.

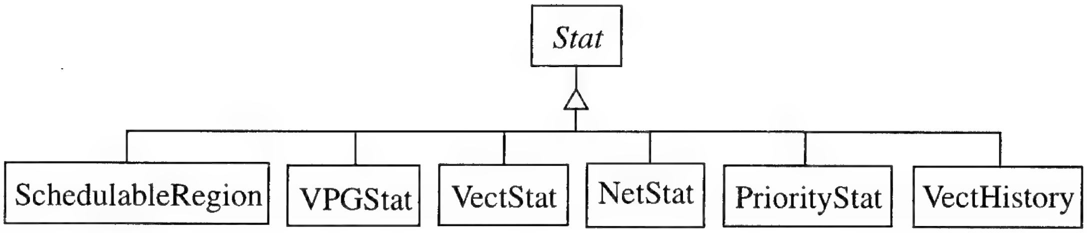


Figure 40 Inheritance relation of statistics classes

Statistics objects are allocated by the class *Interface* when the user clicks on the corresponding buttons. Figure 41 shows the allocation relations. Arrows represent allocation, i.e. an object of the type which the arrow points to is allocated by an object of the type which the arrow starts from. A circle at the point of an arrow means that several objects are allocated.

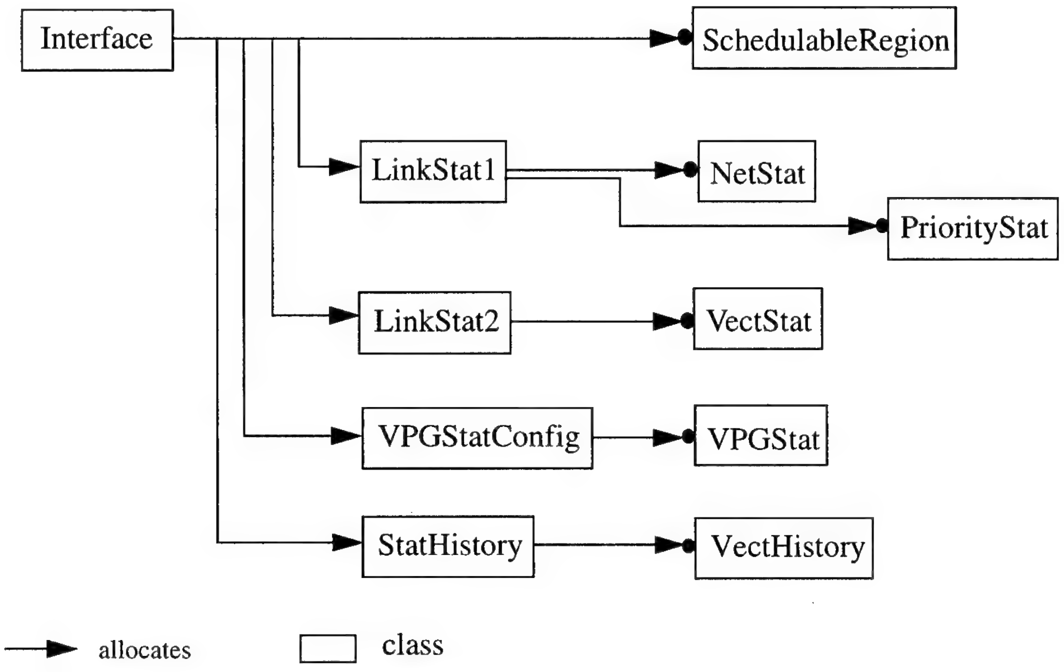


Figure 41 Object allocation

NetStat, *PriorityStat* and *VPGStat* are visualized by cylinders placed on the lines that represent the links. VectStat objects are visualized as floating points in a coordinate system. A *SchedulableRegion* object is shown in the same way, with an additional plane that indicates the range. A *VectHistory* object is visualized as a curve in a 3D coordinate system. The statistics are updated at clock ticks generated by a timer (an OpenInventor object). All the statistics objects are contained in a global array that is managed by the statistics manager. This is shown in Figure 42. By this method, the objects can be updated by traversing the array and calling the update function on each object. The current values that are to be visualized can be found in the shared memory segment for monitoring. Each statistics object has an index which determines the corresponding block in the shared memory segment (see Figure 42 too).

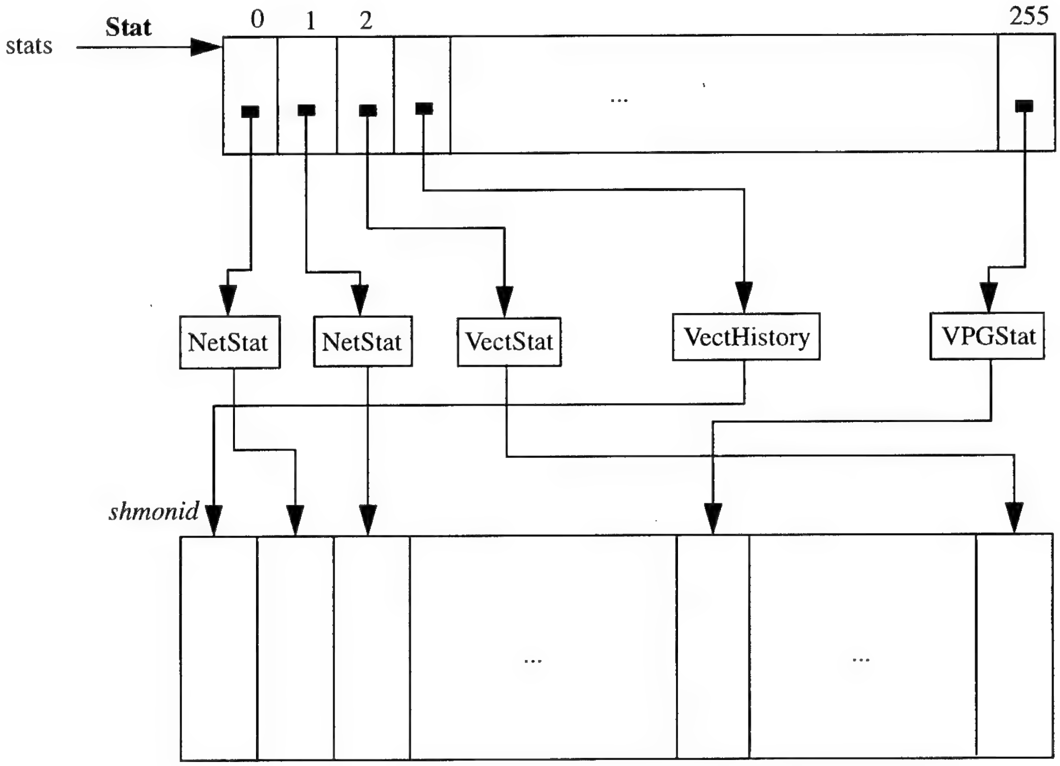


Figure 42 Data structure used to update the bars and diagrams

According to the numerical values the different statistics objects redraw themselves.

5.6.7 User interface and interactive control

The user interface contains beside the mentioned statistics functions management simulation control functions. The emulation control function allows the user to vary the parameters determining call arrival rate and holding time. The simulation control allows the user to change the simulation speed and to monitor the advance of the real clock and simulation time. The network management function encompasses the definition of the adaptivity of the system, the definition of quality of service constraints and the capacity allocation. Other features included are mapping of the relation between the virtual paths and the virtual path groups and changing of the refresh rate for graphics updates.

5.6.8 NoOps

Several functions appear in the user interface, but are not yet implemented. Among these are the robustness control and several functions listed in the pull down menu.

6 Discussion and Summary

6.1 Summary of Work Performed

In this project, we have developed an architectural framework for constructing a virtual private network (VPN) service that enables the transparent interconnection of a private backbone with isolated component networks. The major characteristics of this architecture are:

- The VPN service is based on the concept of Virtual Path Groups and provides a high degree of customer control. As a result, the interaction between customer and provider is minimized.
- The control architecture is structured into three layers of control, according to different time-scales.
- Each control layer is modeled by a generic controller design concept we developed that allows us to implement a large class of control objectives and control schemes.
- A set of performance management capabilities has been realized for this service, including QOS management, VP management and priority management.
- A concept for end-to-end Quality of Service (QOS) in the customer network over a provider network that provides Constant Bit-Rate (CBR) virtual path (VP) has been developed.

The verification and validation of the network architecture is performed on a high-performance emulation platform, specifically, an IBM SP-2 located at Cornell Theory Center. Our approach for validation is based on real-time network emulation, which includes executing the behavior of network components and their inter actions on a high-performance machine. The emulation environment allows us to experiment with the functionality and dynamics of virtual networks (and the underlying transport networks), with greater flexibility and lower cost than implementing components on a real testbed. The emulation system can be controlled by and visualized on an Indigo2 workstation at Columbia connected to the supercomputer over high-speed NYNET links.

6.2 Related Accomplishments

A number of publications were the results of the work performed. These publications are listed below:

[1]. Mun Choon Chan, Hisaya Hadama and Rolf Stadler, "An Architecture for Broadband Virtual Networks under Customer Control," IEEE Network Operations and Management Symposium, (Kyoto, Japan), April 1996.

[2]. Mun Choon Chan, Giovanni Pacifici and Rolf Stadler, "A Platform for Real-Time Visualization and Interactive Simulation of Large Multimedia Networks," 4th Int'l Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), April 15-16, 1996, Honolulu, Hawaii.

[3]. Mun Choon Chan, Giovanni Pacifici and Rolf Stadler, "Prototyping Network Architectures on a Supercomputer," Fifth International Symposium on High Performance Distributed Computing (HPDC-5), (Syracuse, NY), August 1996.

[4]. Mun Choon Chan, Aurel A. Lazar and Rolf Stadler, "Customer Management and Control of Broadband Virtual Networks," IFIP/IEEE International Symposium on Integrated Network Management (IM '97), (San Diego, California), May 1997.

A working prototype of the emulated system was shown in the demonstration program of the Fifth International Symposium on High Performance Distributed Computing (HPDC-5), which was held in Syracuse, NY, from August 5 to August 9, 1996.

6.3 Future Work and Discussions

We are currently working on a public domain release of the parallel simulation kernel used in the emulation platform. This version of the parallel simulator is totally independent of the virtual network architecture and can be used as a general purpose parallel simulator. As the simulator is based on MPI, it is very portable and can run on at least 4 popular UNIX platforms, namely: Sun Solaris, SGI IRIX, HP-UX and IBM AIX. In addition to the features described in Chapter 5, it includes useful features like run-time creation of new object instants and simulation events. The simulation kernel can also be externally controlled such that the simulation can be halted and restarted anytime.

From our experience, we found the idea of prototyping very useful. We have explored this idea in the case of the virtual private network architecture where the control architecture is first prototyped on the emulation platform follow by moving the prototype to the target platform (*xbind* [LAZ96, CHA96d]). Initial experience is encouraging.

In order for a VPG-based VPN to be realized as a viable commerical service, an important requirement is that policing of customer traffic, which is performed per VPG, is done not only at the gateway between the CPN and the VPN, but is also done at the transit switches within the provider domain. Since VPG is a logical concept that might not be recognized by all the switches, additional work is needed. One possibility is to assign a group of VPI's to a single VPG group. For example, all the VPs in the same VPG group can have the same most significant 9 bits in the VPI field and only the last 3 bits in the VPI header can vary (which will allow 8 VPs per VPG). Policing is performed per VPG by masking the last three bits of the VPI header. VP switching remains the same. Such a scheme could conceivably be implemented on existing ATM switches with only minor modifications. A related problem is the requirement that all public network providers agree on the same masking scheme when the VPN covers more than one public network.

References

[ABD90] R. Abdi, R.A. Skoog, “Signalling System No. 7: A tutorial,” IEEE Communications, Vol. 28, No, 7, July 1990, pp. 19-35.

[AND91] G.R. Andrews, “Paradigms for process interaction in distributed programs,” ACM Computing Surveys, Vol. 23, No. 1, March 1991, pp. 49-90.

[AND94] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, LAPACK Users' Guide, Second Edition, SIAM, Philadelphia, PA, 1994.

[ANE96a] Aneroussis, N. G. and Lazar, A.A., “Virtual Path Control for ATM Networks with Call-Level Quality of Service Guarantees”, Proceedings of the IEEE INFOCOM' 96, San Francisco, CA, March 1996.

[ATS93] T. Aoyama, I. Tokizawa, K. Sato, “ATM VP-Based Broadband Networks for Multimedia Services,” IEEE Communications Magazine, April 1993, pp. 30-39.

[BLA90] D. Black, “Scheduling Support for Concurrency and Parallelism in the Mach Operating Systems,” IEEE Computer, Vol. 23, No. 5, pp. 35-43, May 1990.

[BLA95] T.D. Blachard, T.W. Lake, “Distributed Simulation with Locality,” in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 195-198, June1995

[BOL91] R. Bolla, F. Davoli, “A Two-Layer Optimization Structure for Access Control and Bandwidth Sharing in High-Speed Integrated Networks,” Telecommunication Services for Developing Economies, pp. 317-328.

[BOY93] J. Boykin, D. Kirschen, A. Langerman, S. LoVerso, *Programming under Mach*, Addison Wesley, 1993.

[CHA93] Arthur Chai and Sumit Ghosh, “Modeling and distributed simulation of a broadband-ISDN network,” *IEEE Computer*, vol. 26, pp. 37–52, September 1993.

[CHA96a] M.C. Chan, H. Hadama and R. Stadler, “An Architecture for Broadband Virtual Networks under Customer Control,” IEEE Network Operations and Management Symposium, (Kyoto, Japan), April 1996.

[CHA96b] M.C. Chan, G. Pacifici and R. Stadler, “Prototyping Network Architectures on a Supercomputer,” Fifth International Symposium on High Performance Distributed Computing (HPDC-5), (Syracuse, NY), August 1996.

[CHA96c] M. C.Chan, G. Pacifici and R. Stadler, “A Platform for Real-Time Visualization and Interactive Simulation of Large Multimedia Networks,” 4th Int'l Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), April 15-16, 1996, Honolulu, Hawaii.

[CHA96d] Mun Choon Chan, Jean-François Huard, Aurel A. Lazar and Koon-Seng Lim, "On Realizing a Broadband Kernel for Multimedia Networks," Third COST 237 International Workshop on Multimedia Telecommunications and Applications, Barcelona, 25-27 November, 1996.

[CHA97] Mun Choon Chan, Aurel A. Lazar and Rolf Stadler, " Customer Management and Control of Broadband," IFIP/IEEE International Symposium on Integrated Network Management (IM '97), (San Diego, California), May 1997.

[CLA92] D.D. Clark, B.S. Davie, D.J. Farber, I.S. Gopal, B.K. Kadaba, W.D. Sincoskie, D.L. Smith, and D.L. Tennenhouse, "An overview of the AURORA gigabit testbed," Proceedings of the INFOCOM 1992, (Florence, Italy), May 1992.

[COM94] Special issue on Network level modeling and simulation, IEEE Communications, Vol. 32, No.3, March 1994.

[CON95] J.L. Connell, L.I. Shafer, Object-oriented rapid prototyping, Prentice Hall, 1995.

[DUP90] A. Dupuy, J. Schwartz and Y. Yemini, "NEST, A network simulation and prototyping testbed," Communications of the ACM, Vol. 33, No. 10, pp. 63-74, October 1990.

[DZI96] Z. Dziong, Y. Xiong, L.M. Mason, "Virtual Network Concept and Its Applications for Resource Management in ATM Based Networks, "Broadband '96, Montreal, Canada, April 1996.

[ELW93] A.I. Elwalid, D. Mitra, "Effective Bandwidth of General Markovian Traffic Sources and Admission Control of High Speed Networks," IEEE/ACM Transactions on Networking, Vol. 1, No. 3, pp. 329-343.

[FOT95] S. Fotedar, M. Gerla, P. Crocetti, and L. Fratta, "ATM Virtual Private Networks," Communications of the ACM, vol. 38, no. 2, Feb. 1995.

[FRA92] A.G. Fraser, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, and R.C. Restrick, "XUNET 2: A nationwide testbed in high-speed networking," Proceedings of the INFOCOM 1992, (Florence, Italy), May 1992.

[FRA93] S. Frank, H. Burkhardt III, J. Rothnie, "The KSR1: Bridging the gap between shared memory and MPPs," Compcon '93 Proceedings, pp. 285-294.

[FUJ90] R. Fujimoto, "Parallel discrete event simulation," Communications of the ACM, October 1990, Vol. 33, No. 10, pp. 31-53.

[GAF84] E. Gafni, D. Bertsekas, "Dynamic Control of Session Input Rates in Communication Networks," IEEE Transactions on Automatic Control, Vol 29, No. 10, pp. 1009-1016, 1984.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[GOS93] Kaushik Gosh, Bodhisattwa Mukherjee, Karsten Schwan, "Experimentation with Configurable, Lightweight Threads on a KSR Multiprocessor," GIT-CC-93/37, Technical Report, College of Computing, Georgia Institute of Technology, 1993

[GRO94] W. Gropp, E. Lusk, A. Skjellum, "Using MPI," MIT Press, 1994.

[HIS94] Hadama H., Izaki T.,and Tokizawa I.:"Cost Comparison of STM and ATM Transport Networks," NETWORKS'94.

[HAL95] J. Hall. I. Schieferdecker, M. Tschicholz, "Customer Requirements on Teleservice Management," in IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, 1995., pp. 143-155

[HOF95] R.C. Hofer, M.L. Loper, "DIS Today," in *Proceedings of the IEEE*, Vol. 83, No. 8, pp. 1124-1137, August 1995

[HYM91] J. M. Hyman, A. A. Lazar, G. Pacifici, "Real-Time Scheduling with Quality of Service Constraints," IEEE Journal on Selected Areas in Communications, September 1991

[HYM93] J. M. Hyman, A. A. Lazar, G. Pacifici, "A Separation Principle Between Scheduling and Admission Control for Broadband Switching," IEEE Journal on Selected Areas in Communications, May 1993.

[HYM94] J.M. Hyman., A.A. Lazar, and G. Pacifici, ``VC, VP and VN Resource Assignment Strategies for Broadband Networks", Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, D. Shepherd, G. Blair, G. Coulson, N. Davies and F. Garcia (eds), Lecture Notes in Computer Science, Vol. 846, Springer-Verlag, 1994.

[KHE92] S. Kheradpir, W. Stinson, R. Chipalkatti, and G. Bossert, "Managing the network manager," *IEEE Communications*, vol. 30, pp. 12–21, July 1992

[KLE75] L. Kleinrock, *Queueing Systems*, Wiley-Interscience, 1975.

[LAW94] A. M. Law and M. McComas, "Simulation software for communications networks, the state of the art," *IEEE Communications Magazine*, vol. 32, pp. 44–50, March 1994

[LAZ96] A.A. Lazar, K.S. Lim, and F. Marconcini, "Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture," Journal of Selected Areas in Communications, Vol. 14, No. 7., Sep 1996. pp. 1214-1227.

[LEE96] T.-H. Lee, K.-C. Lai, and S.-Y. Duann, "Design of a Real-Time Call Admission Conttoller for ATM Networks," IEEE/ACm Transactions on Networking, vol. 4, no. 5, October 1996, pp. 758-765.

[LEW95] D. Lewis, S O'Connell, W. Donnelly, L. Bjerring, "Experiences in Multi-domain Management System Development," in IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, 1995., pp. 494-505

[LOG92] M. Logothetis, S. Shioda, "Centralized Virtual Path Bandwidth Allocation Scheme for ATM Networks," IEICE Transactions on Communications, vol. E75-B, no. 10, Oct 1992, pp. 1071-1080.

[LOG95] M. Logothetis, S. Shioda, "Medium-Term Centralized Virtual-Path Bandwidth Control Based on Traffic Measurements," IEEE Transactions on Communications, vol. 43, no. 10, Oct 1995, pp. 2630-2640.

[MCC93 W. McCornick, Y. Park, "Approximating the distribution of the maximum queue length for M/M/s queues," Ed. U.N. Bhat, I.V. Basawa, Queueing and Related Models, pp. 241-261.

[MOU95] Constantina Mourelatou, David Griffin, Panos Georgatsos, George Mykoniatis, "ATM VPN services: Provisioning, Operational and Management aspects, "ICS-FORTH Technical Report No. 148.

[MPI94] Message Passing Interface Forum. MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4), 1994.

[NIC93] D.M. Nicol, "The cost of conservative synchronization in parallel discrete-event simulations," Journal of ACM, 40(2):304-333, April 1993.

[NIC95] D. Nicol, P. Heidelberger, "On extending parallelism to serial simulator," in Proceedings of the 9th Workshop on Parallel and Distributed Simulation, pp. 60-67, June 1995.

[OHT92] S. Ohta, K.Sato, "Dynamic Bandwidth Control of the Virtual Path in an Asynchronous Transfer Mode Network, "IEEE Trans. Comm. Technol., 40, 7, pp. 1239-1247.

[ORD96] Ariel Orda, Giovanni Pacifici and Dimitrios E. Pendarakis, "An Adaptive Virtual Path Allocation Policy for Broadband Networks", in Proceedings of INFOCOM'96, San Francisco, CA, March 24-28, 1996.

[PAC95] G. Pacifici and R. Stadler, "Integrating Resource Control and Performance Management in Multimedia Networks," in Proceeding of the IEEE International Conference on Communications, Seattle, WA, June 1995.

[PIT95] A, Pitsillides, J. Lambert, D. Tipper, "A Multilevel Optimal Control Approach to Dynamic Bandwidth Allocation in Broadband ISDN, "Telecommunication System 4(1995), pp. 71-96.

[RIO62] John Riordan, *Stochastic service systems*, New York, Wiley 1962.

[RUM91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-oriented modeling and design,* Prentice Hall, 1991.

[SCH93] J.M. Schneider, T. Preuss, and P.S.Nielsen, "Management of Virtual Private Networks for Integrated Broadband Communication," in Proceeding of the ACM SIGCOMM '93, pp. 224-237.

[SAY95] T. Saydam and J.P. Gaspoz, "Object-Oriented Design of a VPN Bandwidth Management System," in IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, California, 1995.

[SCH87] Mischa Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis,* Addison-Wesley, 1987.

[STE91] J.S. Steinman, "SPEEDES: Synchronous parallel environment for emulation and discrete event simulation," *Advances in Parallel and Distributed Simulation*, vol. 23, pp 95-103, Jan 1991

[STE95] J.S. Steinman, C.A. Lee, L.F. Wilson, D.M. Nicol, "Global virtual time and distributed synchronization," in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 139-148, June1995.

[STO92] M. Stonebraker, "An overview of the Sequoia 2000 project," Proceedings of the COMPCON '92, (San Francisco, CA), February 1992.

[TAN87] A. Tanenbaum, *Operating Systems: Design and Implementation*, Prentice Hall, 1987.

[TSC95] M. Tschichholz, J. Hall, S. Abeck, R. Wies, "Information Aspects and Future Directions in an Integrated Telecommunications and Enterprise Management Environment," Journal of Network and Systems Management, Vol. 3, No. 1, 1995, pp.111-138

[UNG94] B.W. Unger, D.J. Goetz, and S.W. Maryka, "Simulation of SS7 common channel signaling," *IEEE Communications Magazine*, vol. 32, pp. 52–62, March 1994

[WOL89] R.W. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice Hall, 1989.

[YAM91] T. Yamamura, T. Yasushi, N. Fujii, "A Study on an End Customer Controlled Circuit Reconfiguration System for Leased Line Network," ISINM II, 1991, pp. 383-394.

[ZER92] T.G.Zerbiec, "Considering the Past and Anticipating the Future for Private Data Networks", IEEE Communication, March 1992, pp.36-46

☆U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-61063

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.